

- Internationalization
- Spotlight: OS/2 Library

May/June 1995

PRODUCT REVIEW
Prominare Designer
(a GUI design tool)

os/2 Developer

THE MAGAZINE FOR ADVANCED SOFTWARE DEVELOPMENT

Designing with VisualAge

GUI Basics

Exceptions: the Rule for Robustness

BUYER'S GUIDE: GUI Tools

STRETCHING THE LIMITS OF

GUI DEVELOPMENT

U.S. \$9.95 Vol. 7 No. 3



A Miller Freeman Publication

**WHEN I WANT ADVICE ABOUT OBJECT-ORIENTED
PROGRAMMING, I'LL ASK AN EXPERT.**



SO, ASK US.

You already know reusable class libraries promise a dramatic increase in productivity and efficiency for professional application developers. You also know class libraries created with one C++ compiler can't link with code created by another C++ compiler. And that libraries written in one language can't be used with client code written in another language.

But here's something you might not know: MetaWare and IBM are changing all that.

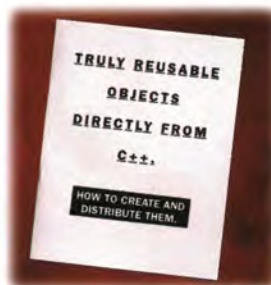
Out With The Old. In With The New. The tight binary coupling that exists between object-oriented class libraries and client code means that changes to either—no matter how small—require a complete recompilation not only of the class itself, but of all the client modules as well.

But with IBM's System Object Model (SOM) and MetaWare's High C/C++ DirectToSOM compiler for OS/2, developers can break that tight binary coupling and extend the advantages of procedure libraries to object-oriented technology.

Pretty amazing, right? And right now, you think this is where our pitch to sell you a compiler comes in. But you're wrong.

Information Only. And It's Free. We know what you really want is information. That way you can make up your own mind about the benefits of SOM and the DirectToSOM High C/C++ compiler for OS/2. We've prepared a white paper that will help. It's called Truly Reusable Objects Directly From C++. How to Create and Distribute Them. It's yours just for the asking.

To receive your free information, call, fax, or e-mail MetaWare at the numbers shown below. Once you review the facts for yourself, you'll understand why MetaWare and IBM mean "objects made easy."



**CALL 408 429 6382
FAX 408 429 9273
OR
E-MAIL TECHSALES@METAWARE.COM
FOR YOUR FREE COPY.
TODAY.**

 **MetaWare®**
OBJECTS MADE EASY.

The Suite Way to Build Portable Applications

NEW!

Is building applications your job? Then life just got easier thanks to the zApp® Developer's Suite for Windows. The zApp Developer's Suite is a set of highly integrated C++ development tools designed to help you transform the blueprints in your mind into commercial quality applications—quickly and easily. And best of all, applications built using the zApp Developer's Suite are portable to fourteen different platforms!

The zApp Developer's Suite consists of zApp, the award-winning portable C++ application framework; zApp Factory™, a fully visual application designer and code generator; and the zApp Interface Pack, a collection of high-level visual objects for the zApp environment. All of these tools are highly integrated to provide maximum ease of use and flexibility.

Rapid Application Development.

Introducing an exciting new visual technology that lets you drag and drop a wide assortment of objects like toolbars, tables, and 3D dialogs; define their characteristics; and build interfaces of any



work, and the zApp Interface Pack, so you have all of their power at your disposal — toolbars, table objects, advanced graphics — in all, over 300 object classes of power just waiting to be tomorrow's best-selling application.

Portability and More.

When you're done building your application, *then* you can decide what platforms you want to support! Applications built using the zApp Developer's Suite are single-source portable to fourteen different platforms. By simply recompiling, your application will run natively on Windows, Win32 (Windows NT, Windows 95, and NT on the DEC Alpha), OS/2®, DOS Text, DOS Graphics and seven X/Motif platforms: IBM RS/6000 AIX, HP HPUX 9.x, SGI IRIX 5.2, SCO UNIX, Sun Solaris 2.x, Sun SunOS 4.1.x, Sun Solaris x86.

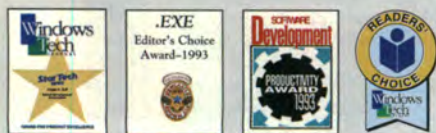
Free Demo.

Sound impossible? Well, if seeing is believing for you, call **1-800-346-6275**, ask for our free demo disk, and get a glimpse of what the future has to offer.

complexity; all in one powerful but easy to use environment. With the click of a button, you can engage a powerful test mode which lets you interact with your application, seeing it exactly like your end user will see it: letting you fill in dialogs, pull down menus, etc. When you are pleased with the look and feel of your application, fully commented C++ source code is only a mouse click away, thanks to the zApp Developer's Suite's code generation capabilities.

Object-oriented Power.

The best news is that this development environment sits on top of zApp, the industry leading C++ application frame-



zApp and Inmark are registered trademarks of Inmark Development Corporation. zApp Factory is a trademark of Inmark Development Corporation. OS/2 is a registered trademark of IBM. All other trademarks are the property of their respective owners.



INMARK

2065 Landings Drive, Mountain View, CA 94043
T: 800-346-6275 or 415-691-9000 F: 415-691-9099

In the U.K. and Scandinavia, call PTS/Software Plus at +44 (0) 928 579900

In France, call PTS/Software Plus at (05) 908194

In Germany, call ESM Software at 07022-9256-0

In Italy, call Silicon Valley On-Line at (049) 654221

In Australia, call Micro Way at (03) 580-1333

BBS: 415-691-9990 • Internet: info@inmark.com

CompuServe: GO INMARK

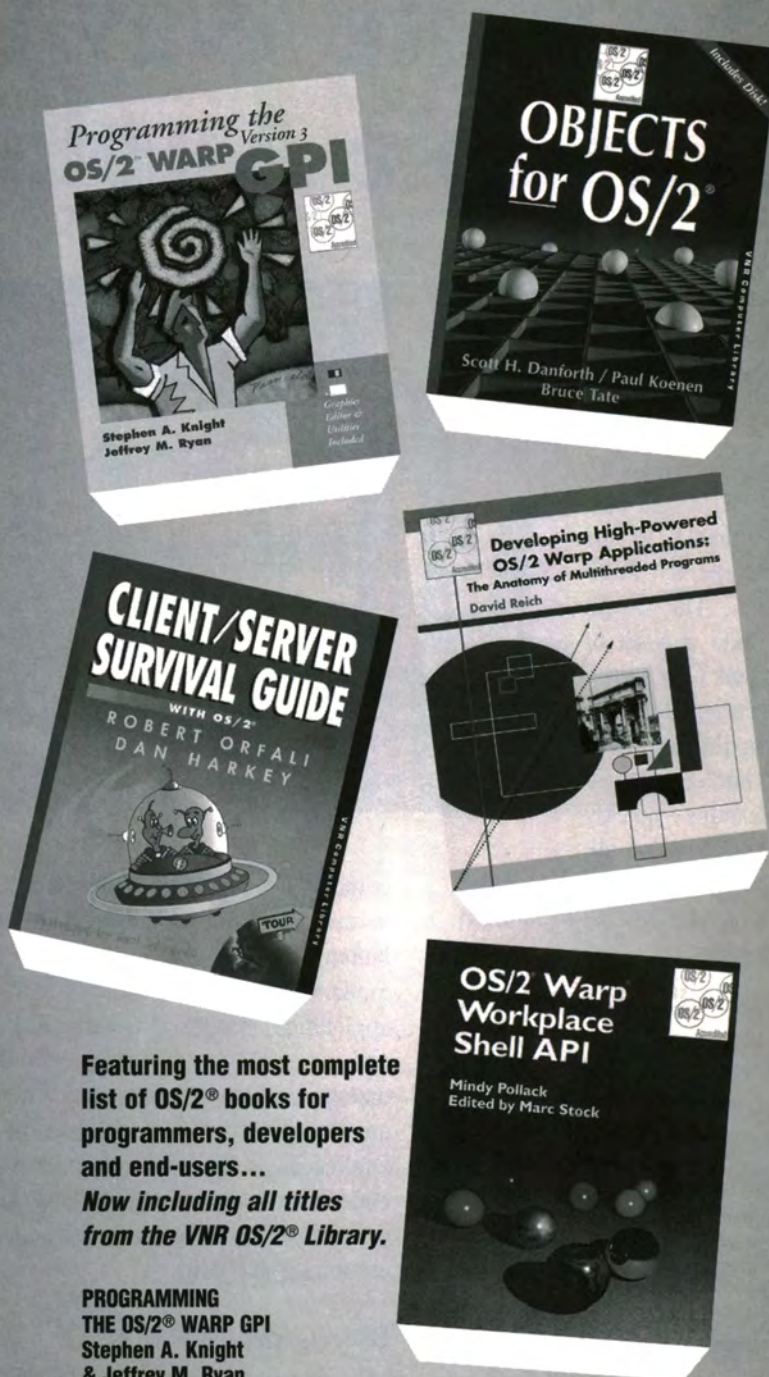
The Bestselling
OS/2® Books
Now Have One Thing
in Common...



They're
Wiley
Computer
Books.

Available now at bookstores
everywhere and from IBM Pub Order.
To order direct, call Wiley Computer
Books at 1-800-225-5945.

For more information,
e-mail compbks@jwiley.com



Featuring the most complete
list of OS/2® books for
programmers, developers
and end-users...
*Now including all titles
from the VNR OS/2® Library.*

**PROGRAMMING
THE OS/2® WARP GPI**
Stephen A. Knight
& Jeffrey M. Ryan
0-471-10718-2
(SR285681) \$39.95

**CLIENT/SERVER
SURVIVAL GUIDE
WITH OS/2®**
Robert Orfali
& Dan Harkey
0-471-13118-0
(SR28549400) \$39.95

OBJECTS FOR OS/2®
Scott H. Danforth,
Paul Koenen
& Bruce Tate
0-471-13126-1
(SR28556800) \$44.95

**DEVELOPING
HIGH-POWERED
OS/2® WARP
APPLICATIONS**
David Reich
0-471-11586-X \$34.95

**OBJECT-ORIENTED
PROGRAMMING
USING SOM
AND DSOM**
Christina Lau
0-471-13123-7
(SR28557000) \$39.95

**OS/2® WARP CONTROL
PROGRAM API**
Marc A. Stock
0-471-03887-3 \$29.95

**OS/2® WARP
WORKPLACE SHELL API**
Mindy Pollack
& Marc A. Stock
0-471-03872-5 \$29.95

**OS/2® WARP
PRESENTATION
MANAGER API**
Marc A. Stock
& Joel Barnum
0-471-03873-3 \$29.95





Programmer's Paradise®

Call for a
FREE
Programmer's
Paradise
OS/2 Catalog!



Gammatech REXX SuperSet/2 by Softouch Systems

The SuperSet/2 gives REXX the power of C by providing interfaces to LAN Server, NetBIOS, TCP/IP and Communications Manager/2, and by providing access to low level system facilities, including processes, threads, semaphores, names, pipes, and VIO commands. The SuperSet/2 complements visual REXX programming packages. The 600+ page manual fully documents over 300 functions in 7 DLLs ready to supercharge your REXX!

Ours: \$69

FAXcetera #: 3621-0002

Paradise #: GTGR33100

key 01MJ95

Prominare Designer by Prominare

A PM programmer's tool for the creation of fully featured GUI's for OS/2. Acting as an extended resource editor, Prominare supports all OS/2 controls for all versions of OS/2 including Warp. Its inherent flexibility enables the power of C to be fully exploited, with the added benefits of intelligent code generation.

Ours: \$495

FAXcetera #: 1017-8601



Paradise #: PRDGO3100

key 03MJ95



Watcom VX•REXX Client/Server 2.1 by Watcom

A visual development environment for OS/2. Powerful connection, query and chart objects allow you to access several databases, manipulate data and chart the results quickly and easily. Features include drag-and-drop programming; bound controls; professional multi-threaded, multi-windowed and drag-and-drop enabled application development.

Ours: \$275

FAXcetera #: 1683-0022

Paradise #: WATVC3100

key 05MJ95

The Object Factory—IDL by Synaptec, Inc.

IDL development environment for OS/2 SOM. Includes multiple inheritance, framework filters, Drag-drop backup and restore, a SOM class browser, on-line help and much more. Use class development notebooks to create new classes just by dropping them on the class browser. Eliminate the complexities of SOM programming. Framework filters focus development providing quicker access to the hundreds of SOM classes. View every inherited method from every parent in one list box. Manage class development with great ease and little effort.

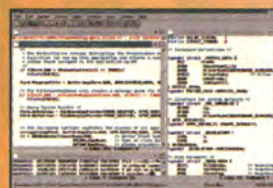
Ours: \$229

FAXcetera #: 1012-5402

Paradise #: SNOFO3100

key 07MJ95

**SOM
Development**



RimStar™ Programmer's Editor by RimStar Technology, Inc.

Features: 32-bit GUI, Syntax Coloring, ANSI "C" macro language, "C" Source Browser, unlimited

redo/undo, no limits on number of files, windows, or line length. Emulates BRIEF plus many other editors. Compile/jump to error, hex editing, smart indenting, bookmarks, completely configurable keyboard mappings, WF/2 enabled and much more. Windows & NT versions available.

Ours: \$259

FAXcetera #: 1013-0901

Paradise #: RSRSP3100

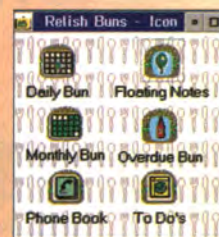
key 02MJ95

Relish by Sundial Systems

Calendar, to do's, and phone book are fully integrated for total reliability, yet also independent for maximum flexibility. Schedule realistically for any time and duration, double-booking as necessary; categorize commitments; repeat events; print schedules; run programs; dial calls; prioritize to do's. Drag-drop, desktop objects ("Buns"), keyword searching, expert system for times/dates. Easy, convenient, CUA compliant, OS/2 Certified.

Ours: \$89

FAXcetera #: 1015-9901



Paradise #: SSREL3100

key 04MJ95



c-tree Plus® by FairCom

DOS • WINDOWS • NT • UNIX • OS/2 • SUN • RS6000 • HP9000 • MAC • QNX • BANYAN • SCO. This well known, highly portable data management package has become established as the tool of choice for commercial development. Offering unprecedented data control, choose from direct low level access, ISAM level, or SQL access with the FairCom Server. Single User, Multi User, or optional Client/Server, ANSI Standard. Full Source. No Royalties.

Ours: \$495

FAXcetera #: 1381-0004

Paradise #: FACT+3100

key 06MJ95

To order call: 800-445-7899
Corporate Developer Division: 800 441-1511
FAX: 908 389-9227
International: 908 389-9228
Customer Service: 908 389-9229
Programmer's Paradise Deutschland:
Tel.: 08121/79073 • FAX: 08121/76566
Programmer's Paradise Italia:
Tel.: 39-2-967-00409 • FAX: 39-2-967-02855
Programmer's Paradise UK:
Tel.: 0161 7284177 • FAX: 0161 7284017
For more information on the products featured on these pages call
FAXcetera: (908) 389-8173

Programmer's Paradise®

1163 Shrewsbury Avenue
Shrewsbury, NJ 07702-4321

• Prices subject to change without notice.
• Call for shipping charges/return policies.

Circle Reader Service Number 4

9
9
8
7
5
4
0
0
8

EDITOR

Dick Conklin
Compuserve:
76711,1005

MANAGING EDITOR

Marta Dils
mdils@mfi.com

EDITORIAL ASSISTANT

Deborah Sommers

EDITORIAL ARTIST

Peter Altenberg

VICE PRESIDENT/**GROUP DIRECTOR**

Regina Starr Ridley

PUBLISHER

Peter Westerman
pwesterman@mfi.com

ADVERTISING SALES

Yvonne Labat
(415) 905-2353
Christian O'Brien
(212) 626-2322
Kristin Morgan
(212) 626-2498

MARKETING MANAGER

Susan McDonald

**MARKETING GRAPHIC
DESIGNER**

Shari Flack

ADVERTISING PRODUCTION**COORDINATOR**

Glenn Sadin

CIRCULATION DIRECTOR

John Rockwell

CIRCULATION MANAGER

Stephanie Blake

SUBSCRIPTION INQUIRIES

U.S.: (800) WANT-OS2
Foreign: (708) 647-5960

CHAIRMAN OF THE BOARD

Graham J.S. Wilson

PRESIDENT/CEO

Marshall W. Freeman

EXECUTIVE VICE PRESIDENT

Thomas L. Kemp

VICE PRESIDENT, PRODUCTION

Andy Mickus

VICE PRESIDENT, CIRCULATION

Jerry Okabe



BY DICK CONKLIN

Editor's Comments

Dragging and Dropping

OS/2's Graphical User Interface support is the best in the business. The Common User Access mouse and keyboard standards and the ever-evolving Workplace Shell give the user a logical, helpful, and friendly way to move around and get things done. The rest is up to the applications.

Fortunately, the OS/2 GUI development platform is equally rich. Whether you write native Presentation Manager applications or use one of the excellent visual tool products, some great OS/2 applications are possible. In this issue of OS/2 Developer, we offer some suggestions. Our authors share their tips and techniques on some basics of GUI: prototyping a new application, writing to the API, and, most important, designing an interface that is right for the intended audience. While few of us have had to develop an online reservation system, it is a great case study for the GUI programmer (see Mark Gayler's article).

See you at the IBM Technical Interchange in New Orleans. Laissez les bon temps roulet!

Following is a letter to Lou Miranda regarding his article in our March/April 1995 issue:

Dear Mr. Miranda,

In the closing comments of your article "OS/2 Programming for the Windows Guru" (OS/2 Developer Vol. 7 No. 2), you state, "Don't be tempted to use OS/2's memory allocation features..." and suggest the use of

malloc() from the standard C library. While, admittedly, this does provide better portability, it eliminates consideration of a very powerful OS/2 (and Unix) concept: Shared Memory. (I have been told Windows provides shared memory as well, but I don't do Windows!)

Thanks for an entertaining article. Except for this one minor statement, I found it quite interesting.

Jonathan McGirr

Chief System Architect

Sterling Software, Banking System Division

Jonathan_McGirr@bsd.sterling.com

Dear Mr. McGirr,

Thanks for your recent letter and kind words regarding my OS/2 Developer article.

You are absolutely correct that you must use OS/2's native memory API functions (such as DosAllocSharedMem()) if you wish to use shared memory. As my article stated, there are exceptional cases where it makes sense to use OS/2 API functions instead of the standard C/C++ malloc() and new(), and this certainly is one. However, due to a lack of space, I gave only one example (allocated but uncommitted memory) and couldn't list all of them, such as your example.

Thanks again for your comments.

Lou Miranda

73567.471@compuserve.com



Dick Conklin

OS/2 Developer: Vol. 7, No. 3, May/June 1995

Subscription information: U.S.: One year (six issues), \$39.95. Canada, Mexico, and international surface mail, add \$16 per year for postage. Canadian GST no. 124513185. International air mail, add \$30 per year for postage. Foreign orders must be accompanied by payments in U.S. funds. For new orders and customer service, call (800) WANT-OS2 (800-926-8672) or (708) 647-5960 (fax: 708-647-5972). IBM employees and branch office customers can subscribe to OS/2 Developer through IBM Mechanicsburg's Systems Library Services (SLSS) using OS/2 Developer's order number: G362-0001. POSTMASTER: Please send address changes to OS/2 Developer, P.O. Box 1079, Skokie, IL 60076-8079. Allow eight weeks for subscriptions to begin. OS/2 Developer (ISSN 1073-0729) is published bimonthly by Miller Freeman Inc., 600 Harrison Street, San Francisco, CA 94107, (415) 905-2200. Application to mail at second-class postage rates is pending at San Francisco, CA, and additional mailing offices.

OS/2 Developer has applied for BPA membership.

© Copyright 1995 by Miller Freeman Inc. PRINTED IN THE U.S.A.



Miller Freeman, Inc.
A MEMBER OF THE UNITED NEWSPAPERS GROUP



This issue, our OS/2 Toolbox column reviews a GUI design tool from Prominare Inc. **By GUY SCHARF**

Prominare Designer Release 4



Guy Scharf

Prominare Designer Release 4 is a GUI design tool for designing the windows, dialogs, menus, and other resources of an application. It generates the resource script and skeleton code for an application. Running on OS/2, Prominare Designer will generate code and scripts for any version of OS/2, Windows 3.1, and Windows NT. You can select C or C++ and can use the CommonView, IBM ICLUI, Borland OWL, or ObjectPM class libraries. For OS/2, you can design an application as a Presentation Manager or Workplace Shell application.

You can import existing OS/2 or Windows resource files. A Presentation Manager Control Extension (PMCX) feature allows you to use Prominare Designer to design applications using your own controls.

An extensive set of rules controls source code generation. You can modify the rules to tailor the standard module and function headers, select which Presentation Manager messages are processed, and define the processing code for each message. You can set the default prefixes for generated symbols to match your naming conventions.

The manual states that 8 MB of memory is required and 16 MB is recommended. The product requires 10 MB of disk space with source code examples or 8 MB without the examples.

I tested C language generation for OS/2 and briefly examined ICLUI code. I did not explore other platforms or class libraries.

USING PROMINARE DESIGNER

When you first start Prominare Designer, it asks whether you want to start a new *design*

or work on an existing design. A design consists of one or more windows and dialogs, which are reflected in a resource script. Figure 1 shows the dialog presented for a new design. Typical of Prominare Designer screens, this form includes many buttons and boxes, only a few of which need to be set. For a new design, the one required field is the "Basename" entry field. Here you enter the name of your project. This name will also be used as the filename stem for the include file and for the source file that will contain the `main()` program. You can ask for the resource script to be generated as an .RC file for use with the OS/2 toolkit or Borland resource compilers, or as a binary .RES file. Other radio buttons allow you to specify the language, target environment, and other controls. The notebook allows you to specify the directory in which source files for different platforms are to be placed.

To create a new dialog or window, click on the New Window button on the toolbar or select New... from the Edit menu. Prominare Designer presents you with a selection of common window configurations: a plain dialog window; a dialog window with OK, Cancel, and Help buttons arranged along the bottom or right side; a secondary window; a pop-up menu; a notebook page; a settings notebook; or a set of common dialogs such as a Product Information or a File Open dialog. Select the type of window desired, and it is created, along with any push button controls.

Figure 2 shows the Window Styles dialog presented for each new window. Using this dialog and its subsidiary dialogs, you set window styles, set presentation parameters,

Migrate your code to 32-bit OS/2 NOW!



Delivers



SMART®

Source Migration Analysis Reporting Toolset

*SMART Tool Helps
Migrate Your Code*

*Get a jump on
your competition
... Get SMART!*

*Get One Up Corporation's
SMART Toolset through
The Developer Connection
for OS/2*

Now you can get SMART through The Developer Connection for OS/2.

SMART analyzes YOUR Windows™ code (16-bit or 32-bit), and OS/2® code (16-bit), sizes the conversion effort, and automatically converts the MAJORITY of YOUR code to 32-bit OS/2.

When you have completed your migration, you'll have a state-of-the-art application that exploits all the advantages of 32-bit OS/2, the leading-edge PC operating system available TODAY!

The Developer Connection for OS/2 is a CD-ROM containing lots of great tools for your OS/2 programming. It is updated quarterly, and comes with a newsletter.

How do I get SMART?

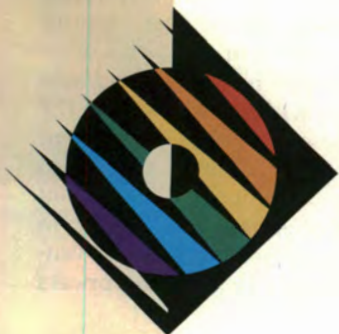
To get the complete SMART Toolset, as well as future versions of the tool, simply join The Developer Connection for OS/2. For more information or to subscribe to The Developer Connection, call 1-800-633-8266 (in Canada 1-800-561-5293).

® IBM and OS/2 are registered trademarks of International Business Machines Corporation.

™ The Developer Connection for OS/2 logo is a trademark of International Business Machines Corporation.

™ SMART—Source Migration Analysis Reporting Toolset is a trademark of One Up Corporation.

™ Windows is a trademark of Microsoft Corporation.





and define any other characteristic of a window that a resource script can specify. Default values for other fields, such as the dialog procedure name, are generated from the window title. If you are generating source code, you need to specify the filename for the source code. Use the Messages... button to say which Presentation Manager messages should be processed in the window procedure. If you need to handle `WM_CHAR` or `WM_MOUSEMOVE`, just click on those entries in the message listbox and the case statement for those messages will be placed in the source code along with skeleton code to handle those messages.

Once you have selected the type of window desired, you place controls on the window in the same way as with OS/2's dialog editor: click on the tool palette or select a control from the Control menu, then click with the mouse to place the control in the window. A dialog for setting the control styles is displayed. The same dialog allows you to select which `WM_CONTROL` notification messages are to be processed. Figure 3 shows the styles dialog for the listbox control.

To resize controls, drag their corners or edges with the mouse. A control alignment button on the toolbar

displays a dialog to align, size, or space controls. To rearrange push buttons from OS/2's along-the-bottom style to Windows' on-the-right-side, select that push button alignment style. To change a control type, such as from an entry field to a spin button, just press the Change Control Types button on the toolbar and pick the new control type.

When your design is done, save it, or press the Generate Code button on the toolbar, and the resource script and application code are written for you. Use your compiler's IDE to generate a makefile and compile the code. To add application logic, edit the generated source code with your favorite editor. You can make further design changes using Prominare Designer; generate the source code again, and your changes to the source code will be preserved.

A common problem for Presentation Manager programmers is that a dialog designed on a high-resolution screen may not fit on a VGA screen or text may be seriously clipped. Prominare Designer provides two aids for these problems. The VGA Screen Template button draws a box showing the VGA 640x480 screen size. The Show Control Limits toolbar button draws a box around all text, showing approximately how the text will be displayed on VGA. As a visual aid, the box is green if the text fits in the control without clipping and red if clipping will occur. Figure 3 shows these marker boxes. The calculation is only approximate, so you must still check operation on VGA before shipping the product. However, the calculation is an enormous benefit during the design process, as it allows you to anticipate and design for other screen resolutions. A Dialog Font Metrics utility will capture font and scaling information for new display types.

Prominare Designer has dialogs for creating and maintaining menus, accelerators, string tables, help tables and subtables, and all other resource script components. It has menu selections to generate help tables and subtables, and it will create a skeleton IPF file from these tables. It will even generate an accelerator table for you from your menu definition—provided you spell the accelerators exactly right in the menu text. According to the vendor, Prominare Designer supports

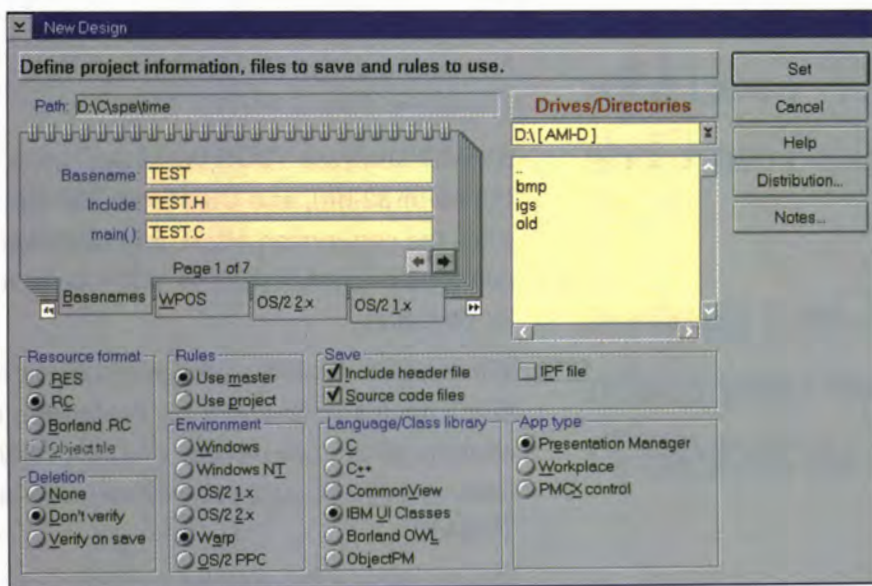


Figure 1. Creating a new design.

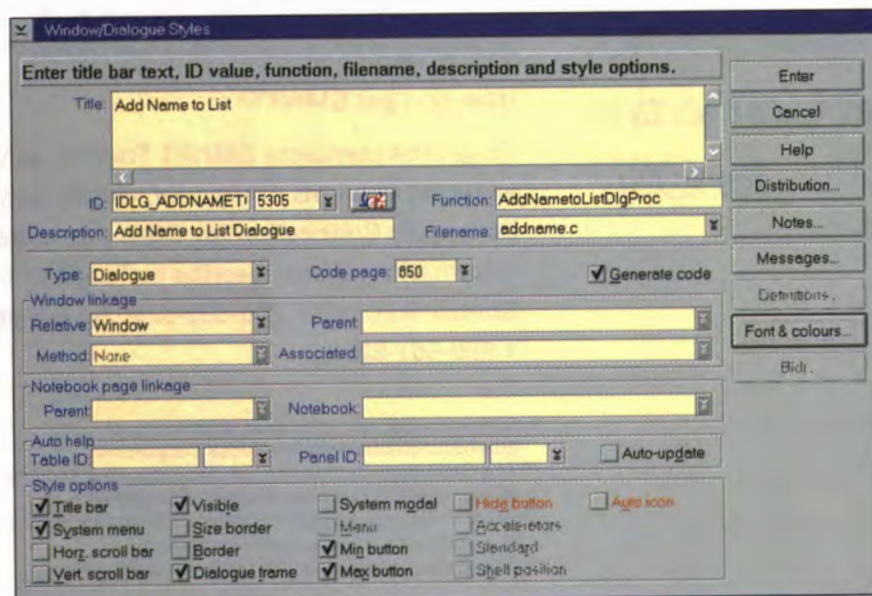


Figure 2. Window styles for a new dialog.



every resource type except RCDATA, RCINCLUDE, FKALONG, and FKAShort.

Prominare Designer is very flexible in importing dialog definitions from RC, RES, DLL, and EXE files and symbols from header files. With the lean application code generated, I have found this tool helpful in altering existing applications—something I would not attempt with other source code generators.

CUSTOMIZING PROMINARE DESIGNER

One of the greatest strengths of Prominare Designer is the flexibility it offers the programmer for customizing its operation. When you select Configure/Creation options..., you are presented with a 21-page notebook that allows you to define the default styles, size, and presentation parameters for each type of control. If it can be specified in the resource script, the default values for controls of that type can be set here. When you later place controls of this type in a dialog, you can modify the styles of that instance of the control as desired.

As an aid in developing or porting multiplatform applications, Prominare Designer presents a superset of all possible styles. Styles unique to OS/2 1.x, OS/2 2.x, and Windows have distinctive colors. This color coding makes it easy to remain aware of any platform constraints of the styles.

Source code generation is governed by a set of *rules*. These rules define every aspect of the source code: contents of the `main()` program, window and dialog module headers, window and dialog procedure outlines, which messages should be included by default, and what code should be included for each message type. Figure 4 shows the rule (skeleton code) for a `WM_ENABLE` message. Buttons on this dialog bring up editing windows for editing the comments immediately following the case `WM_ENABLE`: statement and for editing the program logic up to the `break` statement. You can modify the processing logic for any Presentation Manager message.

Prominare Designer supports both a master set and an application set of rules. When you create a design, you choose to use either the master set or the application set. If you choose application rules, the master rules are copied to the application. Any future changes you make to the rules affect

only the application copy. This approach allows you to have a standard set of rules for your organization while allowing the rules to be tailored to a specific application when required.

Prominare Designer imposes some constraints on the overall program structure. Every procedure consists of a `switch` statement with `case` statements for each message and a `break` at the end of the code for the message. The generator does not support constructs such as placing all `WM_COMMAND` processing in a separate function, but rather it supports only the typical structure with all `case` statements in the window procedure. Since you can change the rules for processing each message, you could make calls to common message processing functions if desired.

CUA compliance is checked each time you change a window or control. Thirteen different CUA issues can be checked; each issue can be selected or deselected in the CUA compliance dialog. CUA checking is too thorough at times: the design tool checks the contents of static text controls, which CUA does not require.

PRESENTATION MANAGER CONTROL EXTENSIONS

Many developers create controls for use in their applications. The PMCX feature allows you to incorporate those controls into the Prominare Designer

design environment. These new controls and their styles are then fully supported in the resource script. Unfortunately, messages and notification codes for PMCX controls are not supported for source code specification and generation.

Implementing a PMCX control requires placing the control in a DLL and adding three DLL entry points (`XxxxRegister`, `XxxxQuery`, and `XxxxStyles`) that tell the designer how the control works and what its styles are. A fairly complex data structure is passed between the designer and the control to communicate all of the options. Once this interface is included, a custom control can be used just like any of OS/2's controls. To add a color wheel to a dialog, just select the custom control tool, pick the color wheel control from the list, and place it on the dialog like any other control.

Prominare Designer includes 17 PMCX controls, all with full source code. These range in complexity from three-dimensional line and text fields to date and time fields, a color wheel, and a notebook with pages. While some of these controls may be directly usable, their only documentation is in the source code for the controls.

DOCUMENTATION

Prominare Designer includes a 340-page manual. About half of the man-

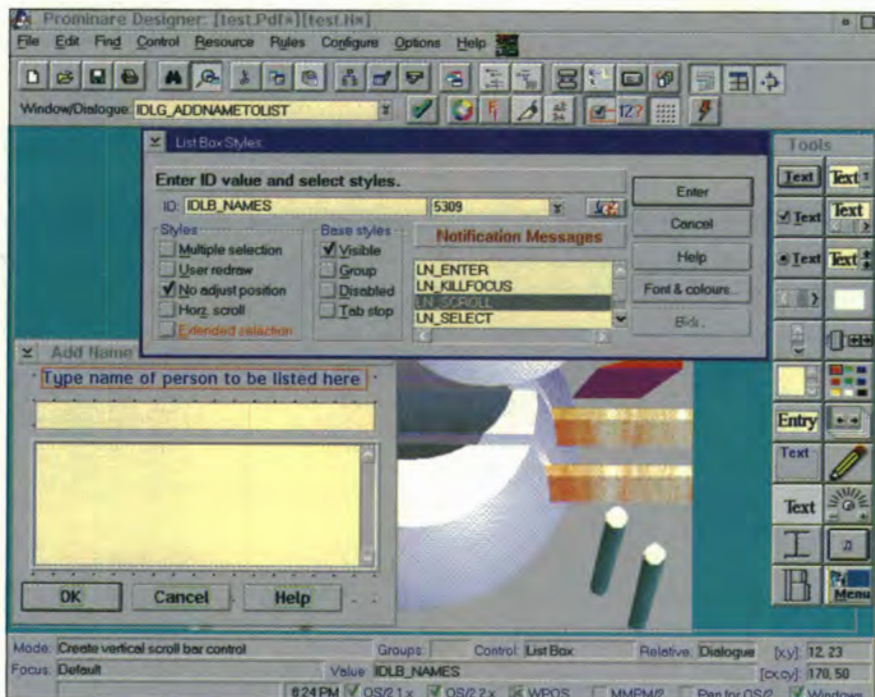


Figure 3. Listbox control styles. Red and green boxes on the dialog predict VGA text clipping.

ual is a reference manual, arranged in order of the menu items. A short tutorial to using Prominare Designer is included. A separate chapter describes the Presentation Manager Control Extensions. Appendixes include a description of CUA compliance checking and comparison of OS/2 and Windows controls. Indexing is poor, making it difficult to find some information in the manual. Learning how to generate help tables and IPF files was a matter of experimentation, as the documentation focused on what you could do, not what you should do.

The tutorial and PMCX sections of the manual are available online as INF files. Online help is provided. Unfortunately, the panel displayed in response to the help button or key is often not helpful. For example, asking for help on the Window Styles dialog results in a help panel saying this is the Windows Styles dialog, without any discussion of the fields in the dialog. An exploration of the help panels reveals that there are more appropriate help panels, complete with hyperlinks to more detailed information. It may be that the help button is displaying help for the menu item instead of for the dialog; if so, that should be fixed. However, since the better panels are not now displayed when needed and the help contents panel is not organized, finding information in the on-line help is difficult.

WARTS AND BLEMISHES

I found Prominare Designer to be stable in my testing. I had one problem with an incorrect resource script generated

with OS/2 Warp selected as the target platform. The vendor quickly provided me with a fix. I also encountered a problem caused by missing data when converting from C to C++ ICLUI. The vendor identified the problem and called me with a solution within a few hours.

Poor online documentation and manual indexing make learning the product much more difficult. There are nooks and crannies of the product that I have not figured out yet.

The dialogs are crowded, with every possible setting usually presented on one dialog. Entry fields and listboxes for symbolic names are often too short for long names. Some listboxes lack horizontal scroll bars, making it impossible to see the full text of a listbox item. If you have changed your default system font to something smaller than the standard 10.System Proportional, the dialogs clip text mercilessly. I had to reset my system font back to OS/2's default setting to read Prominare Designer's dialogs.

The functions you need while designing a window are available either on a convenient two-row toolbar or on the menu. Many of the toolbar options are not available from the menu, so you will need to use both the menu and the toolbar. I was disappointed by the lack of context menus (right mouse button pop-up menus) in the window design environment. You must move to the toolbar or select a menu item for functions that might be reached more easily from a context menu.

Defining a menu is less than ideal, as you have to add items to a list rather than manipulating a visible

menu structure. As usual for Prominare Designer, though, you can set any possible style for a menu item.

LIMITS OF APPLICABILITY

The greatest strength of Prominare Designer is the total control over the resource file and extensive control over generated source code offered the developer. This product could be useful to developers working in C or C++ who want to design in a graphical environment, exploit resource scripts, and generate tailored skeleton code for the application.

You can customize the skeleton code greatly, resulting in code tailored to your organization's or personal requirements and style. However, you can not write your application logic in Prominare Designer. Instead, you must use an editor to add application logic to the generated code. If you later change the design or need to have code generated for additional messages, Prominare Designer will regenerate the source code and try to retain your changes. In my testing, this worked for changes in the window procedures. Changes in the main program and other places were not always retained. In all fairness, the manual does state that certain areas should not be changed, as they are critical markers used in the code generation process. Rather than using special marker lines or comments, Prominare Designer uses the skeleton code in the rules to determine how the source files have been modified. If you delete code from the rules and regenerate the source, the code previously generated may be retained in the source modules.

Prominare Designer is not a rapid application development or visual programming tool. It lacks high-level constructs such as a table display or SQL query tool. Prominare Designer does support all OS/2 control styles that can be specified in a resource script. Additional features that must be implemented by messages are mostly not supported. Thus, you can define notebooks and notebook pages, but you can not put pages in the notebook because that requires procedural code. Similarly, you can define a container control, but making it show a details or other view requires writing application code.

You can add custom controls so the design environment incorporates

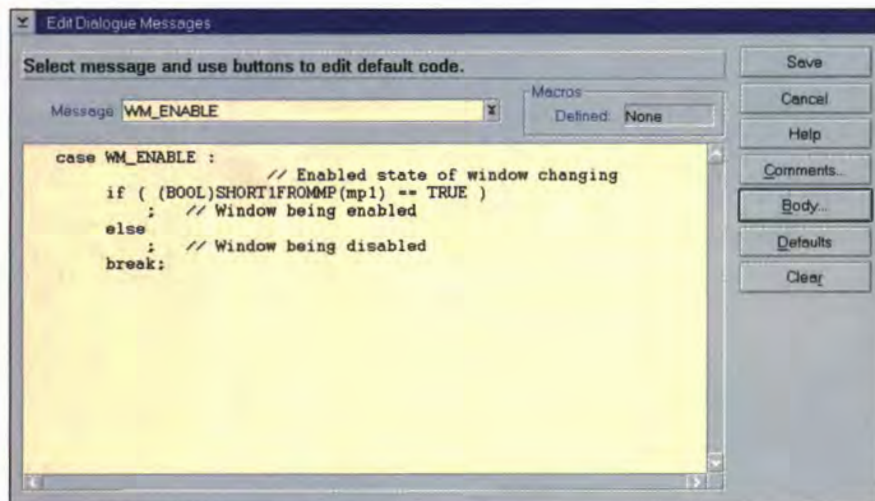


Figure 4. Rule (skeleton code) for WM-ENABLE message.

your custom design elements. You can create objects that have your own functionality or that work around some product limitations. For example, one of the PMCX controls provided with the product is a notebook control that includes the pages in the notebook.

Prominare Designer will generate code for different target platforms (OS/2, Workplace OS, Windows), for different versions on each platform (OS/2 1.x, 2.x, Warp; Windows 3.1, NT), and for different compilers or class libraries at the click of a button. This flexibility may be useful for generating a skeleton application for a different platform or for converting an application from C to a C++ class library.

TECHNICAL SUPPORT

Technical support is available by fax, electronic mail, or in the OS/2 Vendor Forum (OS2AVEN) on CompuServe. OS2AVEN is a recent addition to the company's support offerings, and activity has not built up there yet. My questions submitted via fax or e-mail were answered promptly and completely, often within hours.

Most unusual is that Prominare Designer contains built-in support for documenting and reporting problems and making suggestions. Pull-down menu items take you to notebooks for writing suggestions or documenting problems. After you fill in the pages of the notebook, you can print the form to any printer, save it as an e-mail file, or copy it to the clipboard for pasting into another document or program. With a fax printer, you can print to the fax printer and have the report faxed directly. There is one minor glitch in faxing directly: the fax number is not shown on the dialog, so you have to print the document to see the fax number or look it up in the manual. The vendor has told me that the fax number will be included in the dialog in a future revision so that this process will be even easier.

All suggestions and reports are retained in a database. You can retrieve previously submitted reports to review

or send them again. The program remembers information such as your name and system configuration, so filling out reports is easier after the first one. Including the support mechanism within the product indicates to me a vendor commitment to service and support that I find refreshing.

SUMMARY

Prominare Designer gives you complete control over the resource script and allows you to tailor skeleton code to exactly match your needs. The ability to import resources from other platforms and to save resources and create source code for different targets may help in porting applications. While documentation problems make learning the product more difficult, similarities to OS/2's dialog editor make basic layout and design familiar. Lack of higher-level constructs and support for application code make this a low-level tool for the Presentation Manager application developer rather than a high-level rapid application development or visual programming tool.

Guy Scharf is president of Software Architects Inc., a software development firm specializing in developing OS/2 Presentation Manager applications for vendors and business. He can be reached via CompuServe at 76702,557 or at Software Architects Inc., 2163 Jardin Drive, Mountain View, CA.

PRODUCT INFORMATION

Prominare Designer \$695

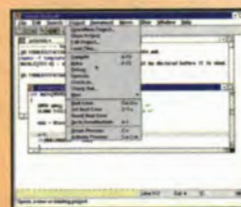
Vendor: Prominare Inc.
222 The Esplanade, Suite 618
Toronto, Ont. M5A 4M8
Canada
Phone: (416) 363-2292
Fax: (416) 363-6157

e-mail: designer@prominare.com
70363,1175 (CompuServe)
OS2AVEN Forum Area 1

Visual SlickEdit®

The Intelligent Programmer's Editor

PC Week's Analyst's Choice '95



OS/2,
Win95,
WINDOWS,
WINDOWS NT

Award Winning Visual SlickEdit is now the standard in programmer's editors. You can completely customize it to suit your programming needs. Program more efficiently using extensive language support, project management, syntax color-coding, and SmartPaste™. Save time using comprehensive CUA, Brief, Emacs, and VI emulations.

Powerful Features:

- ♦ Object oriented C-style macro language and DLL interface
- ♦ Incremental search/search & replace
- ♦ Drag & drop editing
- ♦ Line, column, character selection
- ♦ Spell check comments & strings
- ♦ Clipboard
- ♦ Inheritance™ (patent pending)
- ♦ Undo/redo to 32,767 steps
- ♦ Interactive file compare

MicroEdge also makes SlickEdit available for DOS and 30 UNIX platforms.

Visual SlickEdit supports C/C++, Pascal, FORTRAN, Basic, dBASE, Module-2, Assembly, COBOL, and Ada. The Windows NT version is available for Intel, Alpha AXP, and Power PC machines.

Cut your work in half using:

- ♦ Procedure tagging
- ♦ Syntax color-coding, expansion, and indenting
- ♦ SmartPaste reintends pasted or dropped source code according to nesting level
- ♦ Compiler error processing
- ♦ Macro recording
- ♦ Entire manual on-line
- ♦ Multiple clipboards
- ♦ CUA, Brief, Emacs, and VI emulations
- ♦ Programmable file manager
- ♦ Built in dialog editor
- ♦ Configurable menus and buttons

To order, Call 800-934-EDIT



Also (919) 831-0600

FAX (919) 831-0101

E-Mail 72662.3312@compuserve.com



MicroEdge PO Box 18038, Raleigh, NC 27619

Circle Reader Service Number 6



Continuing the printing theme from last issue, the weighty topic of fonts is tackled.

BY MARK BENGE and MATT SMITH

Fonts 101



Mark Benge



Matt Smith

Type (n) a piece of esp. metal having on its upper surface a character in relief which when inked and brought under pressure against paper leaves an impression of the character.

Font (n) a complete set of types of a particular size and face.

(Taken from the *New Lexicon Webster's Dictionary of the English Language*.)

The process of setting type remained unchanged from the time of Johannes Gutenberg until the late nineteenth century. During that period, type was hand set, meaning each letter composing a word was selected and grouped with other letters placed within a composing stick. As groups of sentences were formed, they were placed in a galley. These galleys would form the paragraphs on the printed page.

This hand setting method was partially replaced by two methods in the late nineteenth century. The first method, Monotype, was based on a machine that could cast the type as individual characters. The second method, Linotype, cast the type as a complete line. These were then placed in galleys that formed the final page.

Unlike hand-set type, Monotype and Linotype were not reused. Hand-set type was based on the reuse of performed type that would be sorted and stored as individual characters. Monotype and Linotype were based on melting down the lead (later a zinc alloy) type for reuse.

This machine casting of type was done by a typesetter sitting at a type of keyboard

that was much like a typewriter (if you have ever seen a Linotype machine, the keyboard layout is not at all similar to a qwerty keyboard). The typesetter would press each key corresponding to the letter needed, the machine would select the mold from its matrix of characters, and the characters would fall into position within a setting bed. Once the line of characters was complete, the molten lead or zinc alloy would be injected into the setting holder and the line of type formed.

Once the metal had cooled sufficiently, the line of type would be removed from the setting bed and placed with other lines of set type. The molds that formed the line of type would then be placed within the sorting mechanism of the typesetting machine to be sorted back into the letter matrix.

It wasn't until the 1960s that typesetting changed. Until this time, type had always been metal based. During the middle of the decade, a new form emerged: phototypesetting. This involved the use of a photographic projection of the type from a film negative onto light-sensitive film or paper. The characters were set at a keyboard in a similar fashion to a Linotype machine except that the keyboard resembled the qwerty layout.

The next major leap forward, with which we are all quite familiar, was digital composition. This arrived in force in the late 1980s with the wide acceptance of PostScript. Here, the font is described in a mixture of raster and vector (mathematical) forms. The final font is then created usually through the use of a laser projected either on a photo-sensitive drum or directly onto film.

Oddly enough, the most advanced development tool for Windows 3.1 isn't from Microsoft.[®]

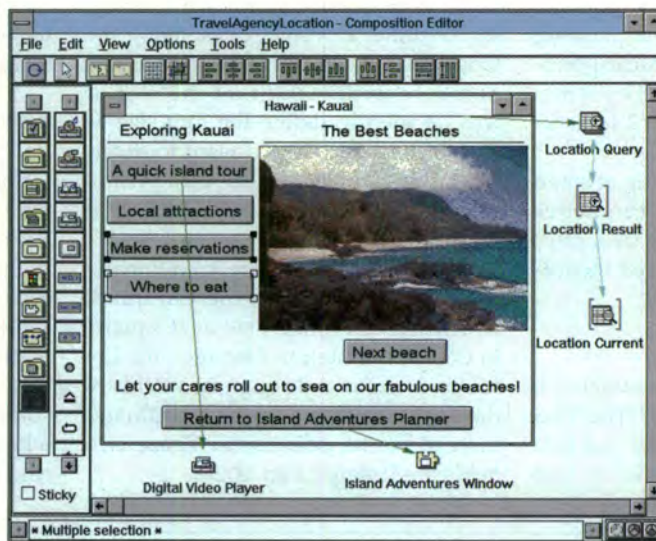
It's never been easier to develop high-powered client/server applications for Windows[™] 3.1.

With VisualAge[™] 2.0, IBM pushes the limits of possibility by combining the simplicity of visual construction with a fully object-oriented development environment.

Mind-blowing multimedia? Awesome apps?

IBM[®] VisualAge 2.0 lets you create powerful and flexible systems with amazing speed.

What's more, you also get the added flexibility



VisualAge gives you the power to quickly create object-oriented applications through the simplicity of visual development.

of intuitive GUI tools and IBM Smalltalk[®].

So now it's easier to develop fully portable, standards-compliant applications in a completely integrated environment.

And you don't have to take our word for it.

IBM VisualAge 2.0 was voted 1994's **Pick of the Year** by *PC Week* and *Datamation* declared it

Best Product of the Year.

For a free 60-day trial copy on CD-ROM, call
1 800 IBM-2279.

*The IBM Smalltalk O-O language, included with VisualAge, is also available separately. Outside the U.S., call (919) 254-4760 or contact your local IBM office. IBM is a registered trademark and VisualAge is a trademark of the International Business Machines Corporation. Microsoft is a registered trademark and Windows is a trademark of Microsoft Corporation. ©1995 IBM Corp. All rights reserved.



WHAT'S THIS GOT TO DO WITH US?

As you can see, major advances in the setting of type were few and far between until the 1960s. Since then, many changes have taken place. Over the last few years, within our personal realm of the computer desktop, we have seen the attached printer go from daisy wheel, to high-end dot matrix, to 300-dpi laser printers, to 1,200-dpi and color laser printers. Correspondingly, we have seen our selection of fonts go from one or two to literally hundreds.

While the technology of using fonts has changed radically, much of the terminology and the concept of how type looks on the printed page has not. What is different is the ease with which it gets there. Okay, that's a loaded statement. The ease depends on your point of view. If you are the user, yeah, easy is the operative word. If you are the programmer, well...

We have to concern ourselves with the relationship of type to our programming so we can better understand how our fonts (basically the fonts that come with our operating systems and printers) relate to the rest of the world. Figure 1 shows the basic terminology used within the printing industry. It also shows the corresponding fields within a font metrics structure that is used within OS/2 to actually describe the fonts.

Yeah, so what? By looking at some of these arcane items, you can better understand how closely (or distantly) the architects of OS/2 followed the traditional use of fonts.

FONT MAGIC

So what are the areas with which we want to concern ourselves? The two most obvious are the font size and face name. These are two of the items that

will bond your application with the user. Most users understand in general terms these two items. To be able to use them in your apps, you need to understand a bit more.

Many of us first muttered unprintable things regarding the baseline and its location. Why is it where it is and not at the bottom of the character? Why didn't those weenies providing the font metrics support stick it in the right place?

Well, they did. The baseline is part of the traditional font description. What the designers of the font support within Presentation Manager have done is provide compatibility not for programmers but for the printing industry.

You can see they have carried the terminology across as much as possible. The only differences are how the point sizes are described and the Em square definition. Generally, we refer to the height of a font in points. For example, this font you are reading is 9-point Palatino.

In the field `sNominalPointSize` in the `FONTMETRICS` structure, the point size is not the actual point size but the size one magnitude higher. Why? Since a font can be defined as being 8.5 points, for example, a problem describing the font arises. So as not to introduce real numbers within the font metrics, it was decided to define the font size as an integer value. If you want to determine the true size of the font within the `sNominalPointSize`, simply divide it by 10.

As for the Em square, traditionally it has been known as the Em quad, which was the point size as a square. In OS/2 Presentation Manager, the Em square is defined through two fields of the font metrics in world coordinates instead of the point size. These two fields are `lEmHeight` and `lEmInc`.

In conjunction with the point size, you need to understand two fields of the font metrics structure that ultimately determine how you interact and select fonts from the system. These are `sXDeviceRes` and `sYDeviceRes`. These two fields do not relate directly to the traditional font descriptions; they are really provided as a means of allowing you to select the appropriate font definition for the device you are targeting.

These fields will contain either the device resolution (Table 1) or 1000. The device resolutions have been defined for CGA, EGA, VGA, and XGA. When a scalable font is being defined, the resolution is 1000. This value along with other information (`FM_DEFN_OUTLINE` flag within the `fsDefn` field of the font metrics) allows you to determine which fonts are device-specific or more global in nature.

SO HOW DO I PICK A FONT?

Generally, you can use the same methodology to select the fonts for a hard copy output device as you would with the display. Some of the techniques for manipulating and scaling the fonts are the same. The real difference is through a selection mechanism, namely, the `lMatch` field of the font metrics structure.

Each font has a unique identifier associated with it, the `lMatch` number. You can refer to this number when selecting a font. But what does this number really mean? Essentially, it is an ID value for the font, nothing more and nothing less. And, to make things just a bit more difficult, the number is the same during the life of the session that is running, but there is no guarantee it will be the same from computer to computer. Therefore, you can not really use this as a sneaky way of referring to fonts.



Figure 1. Printing industry and Presentation Manager terminology.

Where this value is useful is in providing you with a shorthand notation to a font you want to use. You can easily inform the system that you want a particular font by using the `lMatch`. This in effect guarantees that when you find a font you need, such as Helvetica 8 point, you can get it when you use its corresponding `lMatch` value.

Now comes the tricky and not very well documented part about the `lMatch`. Most of us understand that printers are provided with built-in fonts. Since these fonts are designed for the machine, they are generally better in resolution than a font that is downloaded to the printer. So how do you ensure you have selected the printer font and not a system font? By checking that the value of the `lMatch` is negative. Negative? Why negative?

The negative `lMatch` is used by the operating system to indicate that the font selected is specific to the device. This helps to ensure that this font is not selected for a display when a printer device context has been used in picking the fonts.

INTO THE HERE AND NOW

With this basic understanding of the font, along with the magic areas within the font metrics structure, we can begin to apply this knowledge to how we use fonts within our applications and how we actually begin to use them on the printed page. Remember that fonts have been designed for use on the printed page and not so much for the computer screen.

For selecting the proper fonts for the display, we first need to get the screen device context using `GpiQueryDevice`. Then, using this value, we use the `DevQueryCaps` to determine the display resolution so we can match the `sXDeviceRes` and `sYDeviceRes` values of the font metrics.

Using a presentation space for either the window where we want to use the font or the desktop, we request the public fonts. After getting them, we move through the array of font metric structures looking for the font face name, `szFacename`, and the font size, `sNominalPointsize`. We do this in conjunction with checking to see if the `sXDeviceRes/sYDeviceRes` matches the current display device resolution. If we have a match on all accounts, we grab the `lMatch` and use it to select the font. This number is used with the `FATTRS`

structure along with the face name to get the proper font for the window.

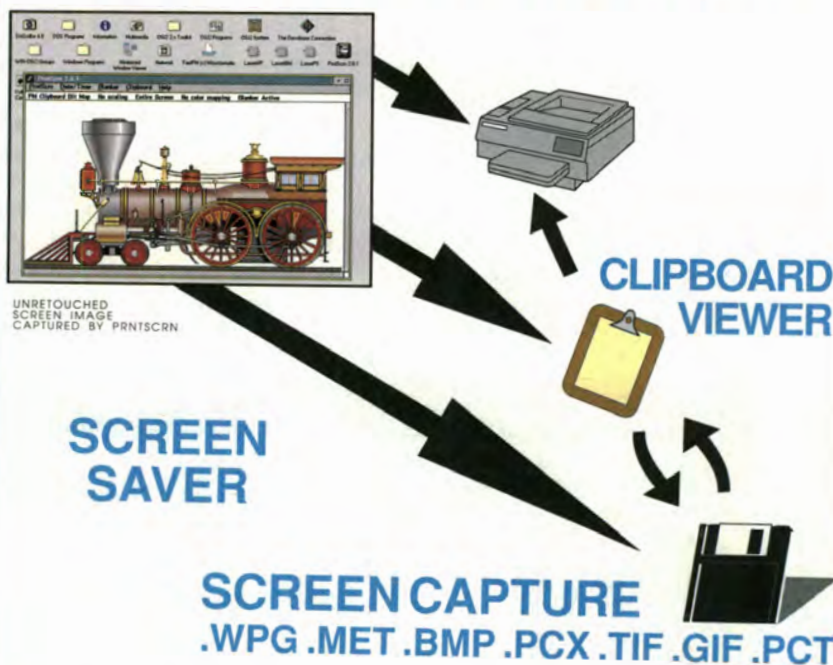
If we can not find the font as a device (bitmap or raster) font, we can re-scan the font metrics array looking for a font that matches the face name and is scalable. Again, we use the `lMatch` to select the font, but this time, we have to do a bit more work to get the font to the appropriate point size. The reason

is that the font is defined within the system as a set of mathematical splines, whereas the device (bitmap or raster) font is defined in the proper size as a bitmap image.

We first call `GpiSetCharMode` to set the character mode to `CM_MODE3`, which allows scaling of the font. After we have calculated the corresponding height of the font in device coordi-



SINGLE KEY-STROKE SCREEN PRINTING



- Quality Color or Black & White copies of color Desktop Images.
- Copy any portion of any image on the Desktop.
- Screen Saver prevents monitor burn-in.
- Date & Time Display for Time-Stamping images.
- LAN installable. Complete support for LAN attached printers.
- Economical! 4 Utilities for 1 Price! Entity licensing available.
- The most stable and reliable product available.
- Includes both OS/2 2.1/2.0 (32-bit) and OS/2 1.3 (16-bit) versions.

THE LEADER: **PrntScrn™** Screen Image Utility for OS/2



TM

MITNOR Software
Phone: (918) 357-1628
Fax: (918) 357-2869

SCREEN IMAGES IN OS/2 DEVELOPER ARTICLES ARE CAPTURED USING PRNTSCRN, COURTESY OF MITNOR SOFTWARE

Circle Reader Service Number 8

Display type	sXDeviceRes	sYDeviceRes
CGA	96	48
EGA	96	72
VGA	96	96
800x600	96	96
1024x768	120	120
1280x1024	120	120

Table 1. Device resolutions.

nates, we scale the font using the `GpiSetCharBox` call.

For the printer, the methodology is very similar, as we mentioned before. The only difference is that you will have to create the printer device context first (perhaps by using the routines from our previous column) and then use the same techniques we described for the display.

THE SAMPLE CODE

Okay class, do you think you can do this on your own? Well, if not, we have pro-

vided the necessary routines as part of our sample code. We have supplied two key functions you can use to select the fonts: `SelectFont` and `SelectScalableFont`. You need to provide only the `hDC` (device context) and the `hPS` (presentation space), and the font will be selected for you.

The method of scaling the scalable font is done through the `ScaleFont` function. The magic here is to provide the necessary point size, which is then converted to the device coordinates.

The function will appropriately scale the font for you from the presen-

tation space. All you have to remember here is that the scaling will remain in effect only while the presentation space is active. Therefore, if you have not created a presentation space through the `GpiCreatePS` function for continual use in the window, you will have to scale the font each time you want to draw with it.

The simple driver routines that illustrate our API usage basically show the selection of the font for use in a window and on your printer.

One thing to think about is how you can improve your applications' handling of fonts. You may want to cache certain font information away once your application has started. We will give you a hint here that you can do this with the DLL initialization if you have your font support within a DLL. You don't have to have a message queue up and running for it to work.

UNTIL NEXT TIME

Now that we know how to select the fonts and use them on the display and

Move Up to the Next Generation!

with

TechBridge Builder®

Create applications by customizing visual component templates supplied by TechBridge.

Link components to external databases by dragging and dropping onto a database class.

Click to generate a quick form, or customize your own. Let our framework worry about SQL, CUA, dialogs, objects, containers, transactions, queries, business graphs, etc. You concentrate on delivering a product that satisfies your users' needs.

Ask about our free demo disk.

Call 1 (800) 463-8998

TechBridge Technology Corp.

5001 Yonge St., Suite 1301, North York, Ontario, Canada M2N 6P6. Phone: (416) 222-8998. Fax: (416) 222-0168.

TechBridge, TechBridge Technology, and TechBridge Builder are registered trademarks of TechBridge Technology Corp. All other product names are trademarks of their respective companies.

Circle Reader Service Number 9

printer, the question that begs to be answered is how do we ensure that the text displayed within the window matches that of the printer for word breaks and such. We will tackle this next time.

If you want a good book that shows some really neat font handling, check out Graham Winn's *OS/2 Presentation Manager GPI* (either first or second edition). It really has some interesting gems in it.

Mark Benge, IBM Software Solutions, Research Triangle Park, N.C., is a staff programmer who joined IBM in 1989 and has worked on various CUA '91 controls for OS/2 2.x. He works in IBM user interface class library development, where he was involved in the implementation of C++ classes for drag-and-drop in C Set++ 2.1. Mark has a B.S. in computer science from Western Carolina University. He can be reached on CompuServe at 73532,2063 and via Internet at banzai@vnet.ibm.com.

Matt Smith, Prominare Inc., Toronto, Ont., is lead architect for the Prominare Development System, an OS/2 2.x advanced GUI development environment. Matt has been actively

involved with OS/2 since 1988 and cofounded Prominare in 1990. He has a degree in architecture from the University of Waterloo. He can be reached via Internet at msmith@interlog.com.



REFERENCES

Credits: screen captures were done through OpenShutter from One Up Corp. and touched up through PaintBrush in Win-OS/2.

Special thanks to Michael Perks of IBM's OpenDoc Development Group for providing insight into the printer objects.

You can download PRNT2.EXE from these electronic sources:

CompuServe's OS2DF2 forum (OS/2 Developer Mag Section).

File area 11 on the IBM PCC BBS, (919) 517-0001

OS/2 Tools section on the OS/2 BBS for IBM TALKLink customers.

FTP to the side address gold.interlog.com and log in as anonymous. The source is located in the /pub/prominare subdirectory.

IF YOU KNOW OS/2®, YOU SHOULD KNOW INDELIBLE BLUE

Power Tools

chartParts	DAK25	\$209.00
Guidelines	JBA53	535.00
KASE:VIP	KAS44	495.00
C++/Views	LIA50	899.00
CA-Realizer	CAS20	89.00
VisPro C & C++	HCK56/55	175.00
SOMobjects Kit	96F8647	255.00
DCE Dev. Toolkit	96F8690	895.00
MKS Toolkit	MKS55	230.00
zApp for OS/2	IMK35	625.00
Object Factory	SAC40	229.00
Objectpm	RAL50/60	210.00
Q+E 5	QES50	344.00
AccessWPS	CRY30	42.00
IPF Editor	PCC84	135.00
Partition Magic		CALL!

Database

DB/2 & DDCS/2		CALL!
OnCmd	ONL40	\$149.00
Watcom SQL	WAT88	295.00

VisualAge



by IBM

IBM's powerful new vision of programming! VisualAge is a client/server application development power tool that focuses on line-of-business applications. Components include visual programming tool, parts library, GUI support, client/server & communication support, enhanced DLL support, multimedia support and much, much more!

17H7495 Single User	
MSRP \$2,495	IB: \$1,550
17H7496 Team V2 OS/2	
MSRP 4,995	IB: \$3,100

Programmer's Editors

PREDITOR/2	COM78	\$149.00
RimStar Prog. Ed.	RIM20	239.00
SourceLink	SSI20	219.00
Qedit	SEM60	69.00
BOXER Text Ed.	BOX10	69.00
SlickEdit	MED10	149.00
KEDIT	MSG50	159.00

Programming Languages

IBM C Set++		CALL!
Watcom C/C++ ³²	WAT22	\$199.00
High C/C++	MET32	524.00
SmallTalk/V	DIG50	895.00
APL2 for OS/2	89G1556	125.00
FORTAN 77 ³²	WAT77	395.00
REXX, REXX & more REXX		
VisPro REXX Gold	HCK20	\$239.00
VX•REXX	WAT10	99.00
REXX SuperSet/2	GTU64	69.00
dbfREXX	DSF25	95.00
REXXLIB	QUR90	45.00

OS/2 and IBM are registered trademarks of IBM Corporation. All other trademarks belong to their respective owners.

THE *single* SOURCE
FOR OS/2® SOLUTIONS

100'S OF OS/2 APPS AVAILABLE TODAY! CORPORATE DISCOUNTS AVAILABLE
CALL FOR OUR FULL COLOR CATALOG ORDERS: 1-800-776-8284



Circle Reader Service Number 10



There are subtle differences when posting and sending messages in Presentation Manager. This issue's *Programming Insider* shows you when and why to post vs. send messages and how to efficiently multithread your programs. **By DAVID REICH**

Hey Mr. Postman



David Reich

Arguably the most important issue in good multithreaded OS/2 programs today is the user interface and how it uses threads. I'm sure most of you have at one time or another heard about the Presentation Manager single queue. Some OS/2 opponents say this means that OS/2 is single threaded. This is not quite true.

OS/2 is a truly multithreaded system and can process many things at the same time, effectively and intelligently moving

things into and out of the processor at lightning speed. Presentation Manager (PM), the windowing and input subsystem of OS/2, is designed to preserve type-ahead and, more precisely, user-input-event-ahead in the system. As such, all user input in the system is funneled through a single system input queue. The input router takes the raw event data (such as keystroke up, keystroke down, and mouse movements) and, based on focus information, mouse location information, and so on, translates it into what you know as PM messages (such as `WM_BUTTON1DOWN` or `WM_CHAR`).

The other job of the input router is to take these generated PM messages and pass them to the application input queues. It is important for message ordering integrity that the messages get to the windows for which they are intended and in the right order. If the focus changes, for example, keystrokes need to be passed to the new focus window. This is the foundation for the design of the single queue.

This is not to say that the design is inherently bad, nor is it to say that it can not or should not be changed. The drawback to this design is that if an application window (or more precisely, the message queue) gets messages from the system input queue but does not respond to them in a timely way, the user interface must hold up processing and distributing more user input events until the receiving window procedure returns. This poses interesting questions in how to handle user input in a PM program. The answer is not that tough.

Your first response (I hope) is to say to do the work on another thread. In fact, in the

```
Initialize critical data structures

WinInitialize

DosCreateThread
DosCreateThread
.
.
DosCreateThread

WinCreateMsgQueue

WinCreateStdWindow

while WinGetMsg
    WinDispatchMsg

WinPostMsg(WM_QUIT) to all threads

Clean Up
Terminate
```

Figure 1. Application initialization.

A few words from our competitors:

Prtblty.
Prtbilty.
Portblty.
Portabit.

No matter how they misspell it, it's still not portability. At Zinc, we understand that porting the last 20% of your code takes 80% of your time.

Only Zinc offers complete portability.

With Zinc® Application Framework™ you can develop on the platform you prefer. And since Zinc is the only one that delivers 100% portability, you'll have your application on other platforms as fast as you can recompile. It's part of what makes Zinc the most productive—and affordable—tool a programmer can use.

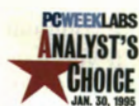
Productivity that leads to opportunity.

Portability is just one road you'll find a little easier. Zinc zips through tedious tasks with C++ object orientation and a unique visual development tool. Globalizing your application is as easy as translating the text. And Zinc is the only product that supplies 100% of the source code.

It all adds up to productivity. Which means more profitability. Which is a concept we probably don't have to spell out for you. For free information and demonstration software, call toll-free:

800 638 8665

Outside the U.S. call: +1 801 785 8900. In Europe call: +44 (0)181 855 9918. In Asia: +81 (052) 733 4301. Contact Zinc electronically at info@zinc.com or GO ZINC on CompuServe.



"Developers seeking easy delivery of GUI applications on DOS, Windows, OS/2, Macintosh and Unix platforms, or pursuing international markets will find Zinc their best option by far."



"Zinc came closest of all the products we tested to our ideal of portability: just copy the code to the target machine, then recompile and relink the application...In short, Zinc did a great job."

z i n c

NO LIMITS


```

void MySecondThread()
{

#define UM_OPENFILE WM_USER+1
#define UM_OTHEROPERATION WM_USER+2
#define UM_OPENFILEDONE WM_USER+3
#define UM_OTHEROPDONE WM_USER+4

HAB hAB;
HMQ hMQ;
BOOL rc;
PQMSG pqmsg;
QMSG qmsg;
HWND hwndSender;

hAB = WinInitialize(NULL);

hMQ = WinCreateMsgQueue(hAB, LONG(0));

pqmsg = &qmsg;

/* Don't forget that since this is a message queue that should not receive */
/* user input and we don't want the system sending us WM_QUIT messages */
/* on shutdown, we need to call WinCancelShutdown */

rc = WinCancelShutdown(hMQ, TRUE);

while WinGetMsg(hAB, pqmsg, NULL, NULL, NULL)

    switch(qmsg.msg)
    {
        CASE UM_OPENFILE:

            open the file
            WinPostMsg(hwndSender, UM_FILEOPENDONE, NULL, NULL);
            break;

        CASE UM_OTHEROPERATION:

            do other operation
            WinPostMsg(hwndSender, UM_OTHEROPDONE, NULL, NULL);
            break;

        default:
            do some default thing in case I got a message I don't know
            what do do with, but I never should...

            break;

        .
        .
        .

    /* End Switch */
    }

    DosExit(0); /* If I got here, I got a WM_QUIT, so end this thread */
}

```

Figure 2. Thread procedure for `WinPostQueueMsg`.

first PM documents I ever wrote, back in 1988, I said the same thing. However, you can do this in different ways. You obviously want to have other threads do the work and free the user input thread to respond to user input events. Should you create a thread for each piece of work as it is required? Should you create multipurpose threads to accept various pieces of work? How should you structure these threads to be event driven instead of having to poll for events? What kind of semaphores should you use? Should you even use semaphores? In *Designing High Performance OS/2 Warp Applications*, I go through all of the rationale and the pros and cons of all of the permutations. I also discuss how to architect and structure the tasks each thread performs. Here I will go into the two techniques I feel are the easiest and the most robust to implement and explain why you need to use them.

UNDERSTANDING ANOTHER THREAD

Whenever a user event comes in, it comes in the form of a PM message. The user interface thread (the one that executes in the message queue and window procedure of your top-level window) gets this message off the queue and, inside `WinDispatchMsg`, calls `WinSendMsg` to have the window procedure process the message. The thread makes a call into the window procedure as a result of the `WinSendMsg` and drops into the `SWITCH` statement to filter the message and act on it.

When the appropriate `CASE` clause is encountered (for that specific message), the window procedure needs to do some work. If it is a simple operation, such as a beep, pulling down a menu, or changing the titlebar, it can be done right in the window procedure. If any significant work needs to be done, you should ship that off to another thread. So what is significant?

Significant work is loosely defined and is more common sense than anything else. If you have a piece of work that could potentially hang, such as communications with another machine over serial or network lines, you should do that on another thread. If you have to do work that could take a long time, such as opening or writing to a file or printing, you should do it on another thread. If you have to update a database or repaint the window,

you should do that on another thread. Essentially, anything that is more than instantaneous should be done on another thread.

WHERE DO THEY COME FROM?

Where do these threads come from? You could, for example, call `DosCreateThread` each time an event requiring significant work comes into your window procedure. The thread would execute and then terminate. Each time your program needs to do work, a thread is started to do it. You could then keep the thread around and not have it terminate in case more work is needed to be done, or you could just let a new one start each time. Once running, you will need to look at what happens when another message requiring the same work arrives or how to notify the main window procedure that a piece of work is done.

These ideas should make you seriously wonder how to do this, with the incredible number of messages passing through PM and the coordination necessary to be able to perform all the tasks reliably. Of course you can create and manage the life of these threads any way you wish, but I will give you my recommendations here.

The first thing is to create the threads at application initialization time. Looking at the pseudo code in Figure 1, you can see that in the main body of the program, you will initialize the required data structures and then call `WinInitialize`. After that, you'll create the worker threads (don't worry about what they'll do just yet, we'll get to that in a moment). After the worker threads are there, you can call `WinCreateMsgQueue` and `WinCreateStdWindow` and begin to process user input in the `WinGetMsg/WinDispatchMsg` loop. Note that you don't create the main thread's message queue until *after* the threads are and just before the main window is created. The reason for this is that as soon as the message queue is there, it is subject to receiving messages. Unless the window is there and a `Get/Dispatch` message loop to receive and dispatch the messages is in place, you would see unpredictable results.

Now that you see how to structure the high-level initialization and how to get the worker threads to exist, let's look at how they run and how you will coordinate execution.

SYNCHRONIZATION

You've seen how to initialize the program and get the threads started, but once they start, what should they do? After all, a thread starts to execute a (set of) function(s), and when the function pointed to by the thread finishes, the thread naturally terminates. There are many ways to manage and coordi-

nate these threads. My preference is to have PM do it for you. You could write the threads to wait on various event semaphores and, when messages come into the main window procedure, have the response in the `CASE` clause trigger the event semaphore to wake up, do the work, and go back to waiting for the next event trigger.



MESATM 2

Spreadsheet Software for OS/2[®]



- Native OS/2 Spreadsheet
- Small • Fast • Powerful
- REXX scripting
- Objects to the core

\$199.
Suggested
Retail Price*

**TO ORDER: CALL
1.800.315.MESA**
or contact your favorite
OS/2 reseller

ATHENA DESIGN

SPREADSHEET TECHNOLOGY FOR A NEW FRONTIER

332 Congress Street, Boston, MA 02210-1217 USA
phone. 1.617.426.6372 • fax. 1.617.426.7665 • email. info@athena.com

*Shipping & handling charges extra. OS/2 is a registered trademark of International Business Machines, Inc.

Circle Reader Service Number 12

However, PM does all of this for you without you having to worry about semaphores. This method also provides you with the facility to make threads multipurpose. It is not something new; in fact, it is something you use in every PM program: the message loop.

When a thread has a PM message

queue, it has the use of PM semaphore handling and synchronization. The way you will use this is that in the code that is each of your worker threads, you will call `WinInitialize` and `WinCreateMsgQueue`, followed by a `WinGetMsg` loop. This gives you the semaphore handling and coordination through PM message passing.

Refer to the pseudo code in Figure 2. You can see that in each thread that will do work on behalf of the application as a result of a message from the main window procedure, `WinInitialize` and `WinCreateMsgQueue` are called to set up the appropriate PM structures. Following that, you see the familiar `WinGetMsg` call. At this point, you have several choices. I will elaborate on two: the use of object windows and the use of `WinPostQueueMsg`. Note that Figure 2 shows what the code would look like only for the `WinPostQueueMsg` method.

OBJECT WINDOWS

An object window is a window whose parent and owner are `HWND_OBJECT`. The properties of this window include the fact that it (and hence, its children) is invisible and as such never receives user input messages. You can structure your code to have window procedures for windows that are children of `HWND_OBJECT`. In the `CASE` clauses of the `SWITCH` statement of their window procedures, this statement does the work you did not want to do in the main window procedure because it executes on the user interface thread. You will define messages yourself using the `WM_USER` values and post these to these object windows.

It is important to post the `WM_USER` messages rather than use `WinSendMsg` because, as you will recall, `WinSendMsg` is a synchronous operation with respect to the originator of the message, while `WinPostMsg` is asynchronous. When posted to the object window, the message arrives on the target message queue, at which time the thread currently blocked on `WinSendMsg` is awakened. This is the semaphore handling PM gives you when using messages.

When the thread is awakened and the message dispatched to the object window's procedure, the `CASE` clauses just described will perform the work based on the user input event that causes the main window procedure to post the message, such as opening or reading from a file or painting some data in the window on the screen.

WinPostQueueMsg

Another method, which is somewhat more efficient, is `WinPostQueueMsg`. `WinPostQueueMsg` operates fundamentally the same way as `WinPostMsg`, but it does not require a window handle. It posts a message directly to a queue.



Using Micro Focus or CA-Realia Workbench to Offload Your Mainframe Development?

X:Change™ is what you've been waiting for!

"In X:Change we found a way to overcome a barrier to programmer productivity and to leverage the latent power in the PC development products we purchased."

- Darrell Harper, USAA

X:Change is the fully-graphical TSO replacement that improves programmer productivity and saves mainframe resources:

✓ Consumes less than one third the resources of TSO. Transfers files over seven times faster.

✓ Full TSO replacement for managing and transferring text and data files, submitting and reviewing batch jobs (JES2&3), even changing security passwords.

✓ Single-step, drag and drop text and data file transfer - PDS(E), VSAM, sequential.

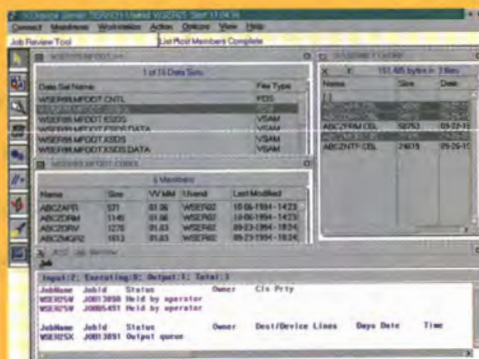
- Record-level data file transfer - sub-set VSAM files on the fly!
- No REPRO, VRECGEN, WFL or reformatting necessary.
- Provides direct integration between PAN/LIB and PVCS.
- SmartCopy only new or changed components, identify dependencies.
- Scan and synchronize mainframe and PC/LAN libraries.

✓ REXX interface and programmable API.

To find out how X:Change makes TSO, IND\$FILE, SDSF/IOF and other MVS utilities things of the past, call today to receive your free copy of "How to Connect" and to arrange a free 30-day trial.

Outside the US and Canada call 415-696-1800
FAX 415-696-1776 • email info@serena.com

800-457-3736



When you use `WinPostMsg`, a window (more precisely, a window handle) must be there to identify the target of the message. `WinPostQueueMsg`, on the other hand, posts the message on the queue without the need of that level of indirection, relieving you of the work (and resource overhead) of extra windows and window handles.

Refer to Figure 2 again, picking up where we left off at the `WinGetMsg` function call. You can see where you would have an "if `WinGetMsg` then `WinDispatchMsg`" for a window procedure. Here we simply call `WinGetMsg` to block if there are no messages and process the `SWITCH` and `CASE` clauses if there are. Since there is no window procedure and no window handle, the work is done right there, eliminating the overhead of another layer of indirection and of creating windows or their associated resource consumption.

This is the fundamental difference between `WinPostMsg` and `WinPostQueueMsg`. Both are asynchronous with respect to the originator of the message, but with `WinPostQueueMsg`, you don't have a window on top of the queue.

For `WinPostQueueMsg` and, actually, any PM thread that does not handle user input directly, it is important to be sure to call `WinCancelShutdown` to ensure that the system does not send that thread a `WM_QUIT` message. If the user asks to shut down the system, PM sends `WM_QUIT` messages to all PM threads. You don't want this to happen to your worker threads since they will all drop out of the `WinGetMsg` loops and terminate in a possibly random fashion. By calling `WinCancelShutdown` for each of these threads, you can control the termination of the threads.

So, to sum up, the general flow is to start up the worker threads early on in the application initialization, each of which will initialize with PM and then block on a call to `WinGetMsg`. The main code then brings up the primary application window and, as processing and time goes on, messages come into the main window procedure and are posted to these other threads either via `WinPostMsg` or `WinPostQueueMsg`. When the processing of each message is complete, the thread posts a message back to the main window that a job was done and goes back to waiting for another message.

By using these techniques, you

can create multipurpose threads and never have to be concerned with semaphores, waking threads up, or when threads should block. Best of all, you can do this off of the user input thread and process all events, user input and otherwise, efficiently and without holding up events in the system input queue.

David Reich has been with the IBM OS/2 development team since 1987. He has worked on many parts of the system, supported customers and application developers, and traveled the world giving seminars and teaching OS/2. His latest book is, *Designing High Performance OS/2 Warp Applications*, published by John Wiley & Sons. He can be reached on CompuServe at 76711,632 or via the Internet at speedracer@vnet.ibm.com.



Quadron development tools for IBM ARTIC cards:

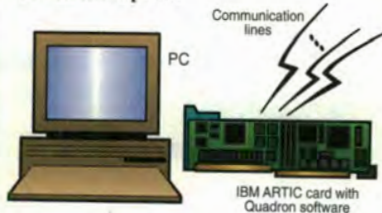


How to communicate better.

Increase system efficiency.

Our development software allows you to shift communication tasks from a PC to an IBM ARTIC co-processor card specially designed to handle them.

You'll gain CPU independence and multiple high-speed ports. Plus, each port on the IBM card can potentially support a different protocol and a different line speed.



Choose just what you need.

We have off-the-shelf solutions for your everyday data transmission needs, and tools for custom situations. You can select from HDLC/SDLC, Bisync, Async, X.25, LAP-B and custom protocols.

Your host PC can be an ISA, EISA, or MCA machine running various operating systems. And possibly most important, you can port co-processor code that you write from one operating system to another with little or no modification.



Serve vertical markets.

Quadron and IBM have teamed up to help people worldwide manage demanding applications such as:

- Travel reservation
- Stock market data delivery
- Telecommunications switching
- Retail point-of-sale purchases
- Process control
- Automatic teller services
- Shop floor control.

Work simpler, work smarter.

Quadron software is easy to install, easy to use, and downright productive. The high-level, C-language API makes co-processor programming simpler. Our user manuals have numerous code examples to speed you on your way. And our support, should you need it, will quickly serve up accurate answers to questions, whether they come in by phone, fax, or through our BBS system.

Act right away.

For today's dependable solution to tomorrow's communications needs, contact us now.



209 East Victoria Street
Santa Barbara, CA 93101 USA
fax 805-966-7630
telephone 805-966-6424

© 1995 Quadron Service Corporation. All trademarks are the property of their owners.

Circle Reader Service Number 14



Presenting a class library for rapid design prototyping with VisualAge. **By ANDY CUBBON**

Object-Oriented User Interface Design with VisualAge



Andy Cubbon

For my user interface design projects, I needed a tool that could quickly prototype the object-oriented user interface (OOUI) or graphical user interface (GUI) designs I develop for my clients. While I still haven't found a tool that is faster than the easel charts I use during my interactive joint design sessions, I now use VisualAge to quickly develop a working prototype after the session. Frequently this can be done in the evening after the workshop and demonstrated to the client the next day.

Actually, building the windows and dialogs is the easy part. Coming up with the sample data and putting it in a test database can take more than half the time. The key to rapid design prototyping is having a library of classes that contain the common window and dialog structures with their associated Smalltalk methods. Over the years, I have found that many common menu choices, dialog windows, and push buttons can be reused from product to product. For example, most products need some kind of print-

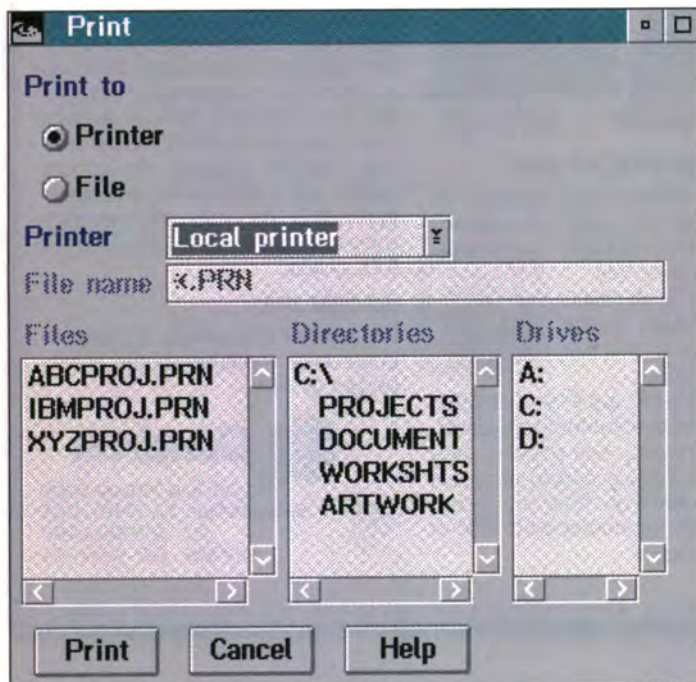


Figure 1. Example of a general-purpose Print dialog window.

```
positionSizeWindowAtX: x atY: y
"Set the initial window position and adjust
initial window size to the current system.

Invoke with #aboutToOpenWidget event."

"Set initial position."

(self partAttributeValue: #(#'Window' #self))
primaryWidget x: x asNumber.
(self partAttributeValue: #(#'Window' #self))
primaryWidget y: y asNumber.

"Set size of development environment screen"

ScreenFunctions developmentScreenWidth: 640.
ScreenFunctions developmentScreenHeight: 480.

"Size the window"

ScreenFunctions sizeWindow:
(self partAttributeValue: #(#'Window' #self)).
```

Figure 2. Method (script) in the `Window` class (part) to set the initial window size.

HURRY! GET THE

OS/2 WARP & STACKER BUNDLE

AT A GREAT LOW PRICE,

PLUS A \$30 REBATE!

OFFER ENDS 7/15/95



Once again, we've stretched the limits of data compression. Because with new Stacker® 4.0 for OS/2 & DOS, you'll now be able to store an average of 2.5 times as much data on your hard disk. And you can do it up to four times faster than with previous versions.

Stacker 4.0 is compatible with OS/2, Windows and DOS applications, and new OS/2 Warp. And if you're running DoubleSpace™, DriveSpace™ or SuperStor/DS, Stacker's conversion program is the easiest way to move to OS/2 without uncompressing your drive.

Stacker AutoSave™ helps put the squeeze on lost data by backing up and protecting vital file system

information. And the new Stacker Toolbox™ gives you instant access to your Stacker tools from your OS/2 desktop—just point and click.

Stacker 4.0 for OS/2 & DOS is now available for substantially less than previous versions and at special upgrade and cross-grade rates for current Stacker users. And we go to every extent to keep you happy with our 60-day, money-back guarantee.

Call 1-800-522-7822, ext. 8603 or contact your local dealer. Then get Stacker 4.0 for OS/2 & DOS, and give your hard drive more room to breathe.



NOTHING EXPANDS YOUR HARD DRIVE LIKE NEW STACKER 4.0 FOR OS/2 & DOS.



Outside of the U.S. call 1-619-794-4333 or fax 1-619-794-4575 for more information.

© 1994, Stac Electronics. Stac and Stacker are registered trademarks, and Stacker Toolbox and Stacker AutoSave are trademarks of Stac. OS/2 is a registered trademark of IBM. MS-DOS, DoubleSpace, DriveSpace and Windows are trademarks of Microsoft. All other product names are trademarks of their respective owners. Novell Yes: Developer tested only. Novell makes no warranties with respect to this product. Actual compression results may vary.

Circle Reader Service Number 15

ing support. Whether the environment supports drag and drop or not, a dialog lets the user select a printer or direct the printing to a file as needed, as illustrated in Figure 1.

ADJUSTING WINDOW SIZE TO SCREEN SIZE

The first design prototype I did took me about 16 hours. This included developing most of the standard win-

dows as well as the product-specific windows. The prototype also had to run on both VGA and XGA displays. I spent most of the time attaching the controls together and developing the methods (scripts)¹ to set the initial window size so the prototype windows would display appropriately at the different resolutions. Now that I have included these methods in the `Window` class of my prototype library, they are inherited by the other classes in the library as well as new design prototype windows using the library classes as parents. All I need to do is specify the initial position of the new window.

Figure 2 is a method in the `Window` class that sets the initial position and size of the window. It sets two class variables of `ScreenFunctions` (a nonvisual part in the prototype library) with the development system screen size and uses the `#sizeAWindow` method of `ScreenFunctions` (Figure 3) to set the initial size of the window to be proportional to the execution environment's screen size.

These methods and the others in this article are available in a file named `CUBBON.ZIP` in the `CompuServe OS2DF2` forum library, `OS/2 Developer Mag` section.

Figure 3 is the method in `ScreenFunctions` to set the initial window size proportional to the screen size. It calculates the ratio of the current system screen width to the development system screen width. This ratio is used to adjust the initial size of the window. The ratios for XGA to/from VGA were determined by experimentation to get the resized window to display well. Note that the `#initialize` method only sets the development system class variables if they have no value (are nil). Since they are set by the method in Figure 2, they are not changed by `#initialize`.

THE PROTOTYPE CLASS LIBRARY

Figure 4 shows the class hierarchy for the prototype library. The `Window` class inherits from the standard `VisualAge` visual class (part), `AbtAppBldrView`. `Window` is an abstract class. Design prototypes do not create instances of `Window` or use it as the parent class. Instead, classes built for the design prototype would inherit from or be instances of the appropriate subclasses of `Window`.

`PrimaryWindow` is used as the parent class for a details view of a container

```
sizeAWindow: aWindow
    "Size the argument window proportional to the current
    screen size based on the development screen size.

    Invoked by the #positionSizeWindowAtX:atY: method."

    | windowWidth windowHeight ratio |

    "Ensure argument is a VisualAge window"

    (aWindow class inheritsFrom: AbtBasicView)
        ifFalse: [self error:
            '#sizeAWindow argument is not a visual part.'].

    "Intialize the class variables if necessary"

    (CurrentScreenWidth isNil)
        ifTrue: [self initialize].

    "Get the Width and Height of argument window"

    windowWidth := (aWindow primaryWidget width) asFloat.
    windowHeight := (aWindow primaryWidget height) asFloat.

    "Calculate screen size ratio. Assumes that the ratio is
    the same for the width and height."

    ratio := ((CurrentScreenWidth asFloat)
        /
        (DevelopmentSystemScreenWidth asFloat)).

    "Adjust ratio based on experiments of VGA to XGA
    and XGA to VGA."

    "VGA to XGA"
    (ratio = (1024 asFloat / 640 asFloat) )
        ifTrue: [ratio := 1.3].

    "XGA to VGA"
    (ratio = (640 asFloat / 1024 asFloat) )
        ifTrue: [ratio := 0.85].

    "Compute new window size based on the ratio of
    the current screen to the development screen"

    windowWidth := (ratio * windowWidth) asInteger.
    windowHeight := (ratio * windowHeight) asInteger.

    "Set the new window size"

    aWindow primaryWidget width: windowWidth.
    aWindow primaryWidget height: windowHeight.
```

Figure 3. This is a class method used to set the initial window size.

window and the main window used to display an object. It is a window with standard menu bar and pull-down choices and standard dialog and secondary windows. The design prototype primary windows are tailored to include the appropriate controls and non-visual classes (parts) for the object or container. For example, a database query and connected table controls are added to the details container view. The appropriate menu choices are created by deleting those that are not needed and adding any design-specific choices.

The **PrimaryWindow** menu choices that display dialog windows use those provided in the prototype library. Therefore, if the prototype library windows meet the design prototyping requirements, no changes need to be made. This includes sample help windows.

If the design requires variations of a prototype library dialog, a new window is created using the appropriate prototype library class as the parent. An instance of the new dialog window is put in the composition area, and the connections are moved from the library instance to the new one. Once the new window works properly, the library instance is deleted.

HELP WINDOW

The prototype help window (Figure 5) is intended to illustrate how the help would work rather than be a true help system. Therefore, to save prototype development time, it does not use the help support provided by OS/2. Nor do any of the standard actions, such as Search or Index, work.

A method in the **Window** class of the prototype library (Figure 6) displays the help window. This method takes two arguments. The first argument is a string used as the title of the help window. The second argument is the name of an ASCII file used by a **HelpWindow** method to provide the content of the help window. The arguments for invoking a help window are typically provided as parameters in the Settings of the event-to-script connection for the help actions. If a file name is not provided (the argument is nil), the method in the **HelpWindow** class (Figure 7) displays the default message shown in Figure 5.

PRODUCT INFORMATION WINDOW

The Product Information window (sometimes called the About window)

illustrates this function of this Help menu choice. It should be tailored to the prototype by including the name of the product and the author of the design prototype. Create a new visual class in the prototype application that inherits from **ProductInformationWindow** and change the appropriate labels and icons. Add the new Product Information window to all primary windows. Drag the connections from the library Product Information window to the new one, test it, and delete the old window.

DIALOG WINDOW

The generic dialog window is used as the parent class for the other standard dialog windows in the prototype library as well as the parent for any application-specific dialogs that are needed by the design prototype. It contains the standard Cancel and Help push buttons along with a push button to perform the main action of the dialog. It also contains the generic example help window, which can be displayed by pressing either the Help

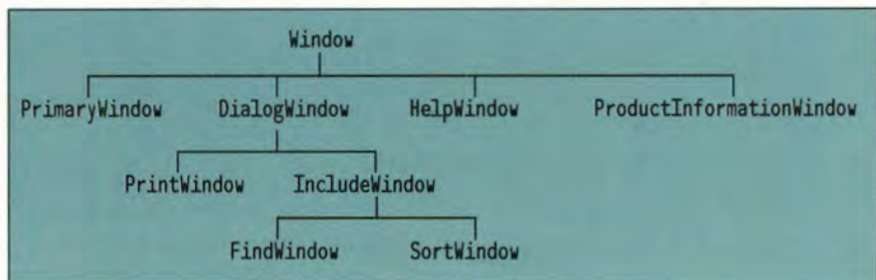


Figure 4. Prototype library class structure.

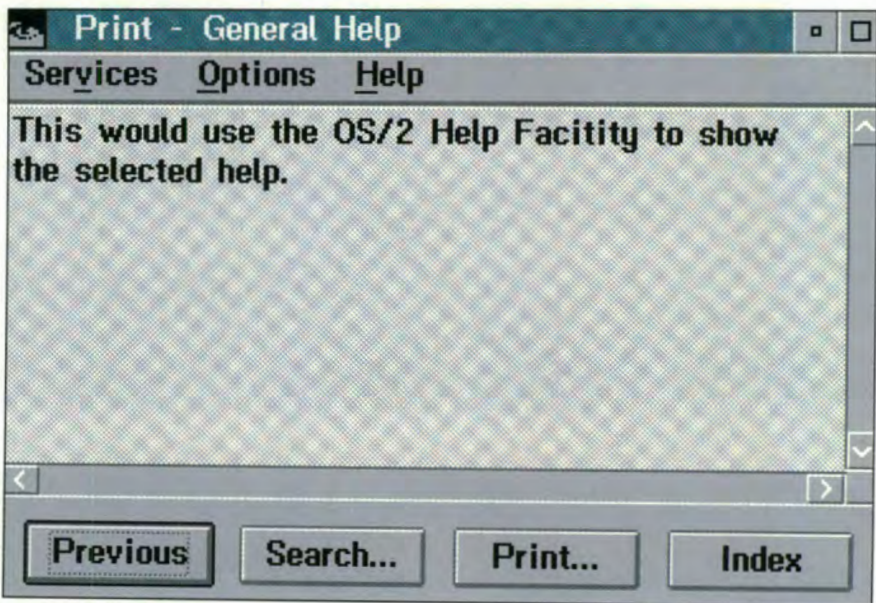


Figure 5. Prototype Help window.

```

helpWithTitle: titleString inFile: fileNameString
    "Display generic Help window."

    self partAttributeValue: (#Window_Help #HelpTitle)
        put: titleString.
    self partAttributeValue: (#Window_Help #HelpFile)
        put: fileNameString.
    (self subpartNamed: #Window_Help) performActionNamed:
        #openWidget.
  
```

Figure 6. Method in the **Window** class used to display a generic help window.

push button or F1.

As with the `PrimaryWindow` class, to create a new dialog window for a pro-

totype, create a new visual class using `DialogWindow` as the parent class. Add the needed controls, change the title and

action push-button text, and even add an additional push button or two if needed.

```
initializeWindow
"Check to see if a file name was passed.
If so, get the help text from it.
If not, use default text.

Invoked by #aboutToOpenWidget."

( ((self partAttributeValue: #(#HelpFile #self)) isNil)
  or: [(self partAttributeValue: #(#HelpFile #self)) size = 0] )
  ifTrue: [
    self partAttributeValue: #(#HelpText #self)
    put:
      'This would use the OS/2 Help Facility to show the selected help.'.]
  ifFalse: [self readTextFromFile].
```

Figure 7. Method in the `HelpWindow` class used to determine if a file name is passed.

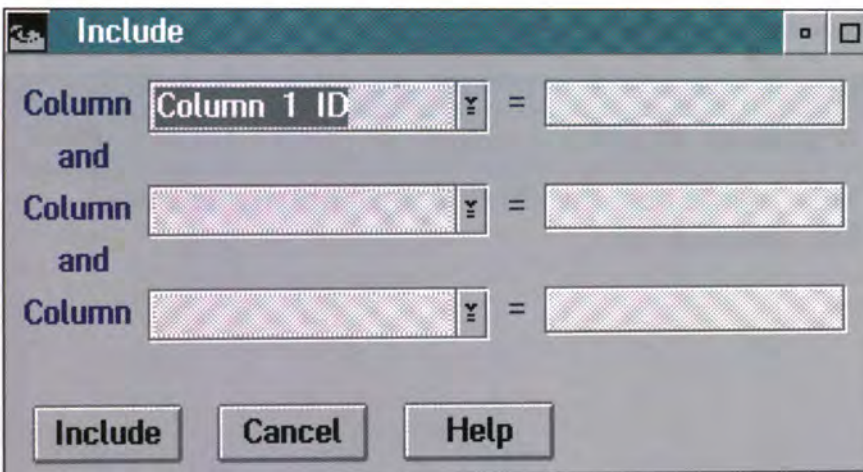


Figure 8. Include dialog window.

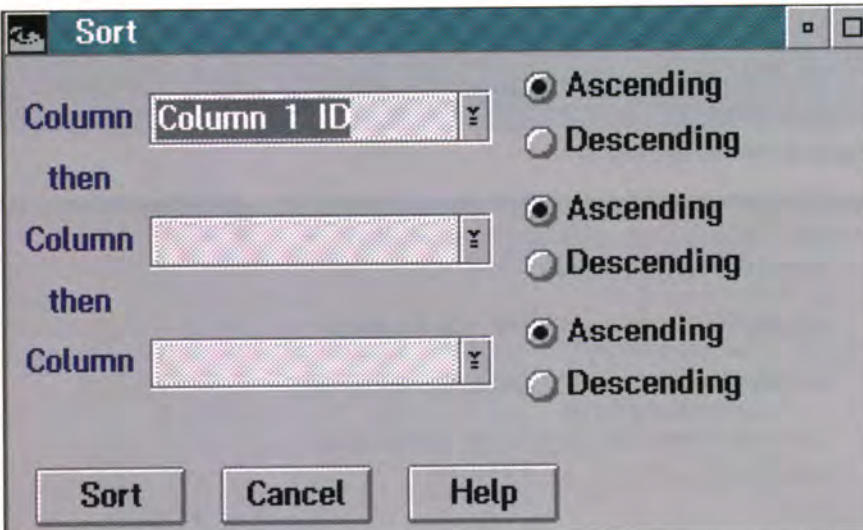


Figure 9. Sort dialog window.

PRINT WINDOW

Figure 1 illustrates a model for a Print dialog window. The radio buttons let the user select where to direct the printing. If Printer is selected, the user can select a printer from the drop-down list. Using a drop-down list lets the product populate the list with the available printers at installation time or even at run time. If File is selected, the user can enter a file name or select one from the list boxes. Only the appropriate controls are active for each radio button.

When the Print window needs to be tailored for a specific design, create a new class as a subclass of `PrintWindow` and make the desired changes. For instance, an additional option for directing the printing to a fax modem can be added. Add another radio button and an entry field for the phone number. Copy the method that interprets the radio buttons from the parent class and enhance it to handle the new radio button.

INCLUDE WINDOW

The Include choice in container windows allows users to filter the objects displayed to include only those that meet certain criteria. Figure 8 shows a simple Include dialog window that lets users specify values for up to three columns as their criteria. The column selection uses a drop-down list that provides users with flexibility in their filtering criteria. It also allows the application to provide the values at run time, enabling this dialog to be used for any container.

The Include dialog can be enhanced to allow users to OR the criteria as well as AND them, for example, "Last name = Smyth" "or" "Last name = Smith." Additional comparison operators such as less than, greater than, and not equal to can be added by using a drop-down list of operators. Increasing the flexibility of the Include dialog adds additional programming effort to the actual product and may be impractical when the criteria is used to generate database queries.

FIND WINDOW

The Find choice in container windows allows users to locate objects in the container. The Find dialog is similar in

appearance to the Include dialog. The only appearance differences are replacing the word Include with Find in the titlebar and first push button.

Again, this window can be made more flexible by adding OR and providing additional comparison operator choices.

SORT WINDOW

Users can also sort objects in a container. Figure 9 is a general Sort dialog window. It is similar in appearance to the Include and Find dialogs by allowing users to select the columns to sort from drop-down lists. Instead of comparison operators and an entry field, it gives the user a choice of sorting in ascending or descending order for each column.

These prototype Include, Find, and Sort windows just illustrate the dialogs in a design prototype of the interface. For the actual application, significant additional Smalltalk programming and database access elements would have to be added to do the actual filtering, locating, and sorting.

FUTURE ENHANCEMENTS

This is beginning rapid design prototype class library in VisualAge. More objects will be added as they are needed in designs. Some candidates under consideration are standard message boxes. Unfortunately, the message box support in VisualAge does not have enough flexibility for all situations. Specifically, I plan to develop a standard message dialog that is displayed when a user does not save changes to an object and takes an action that would cause the changes to

be lost. This message needs Save and Discard buttons, which are not provided by VisualAge message boxes.

I am also planning to develop a standard Look Up dialog that is displayed for selecting valid values for a field. A Look Up dialog is used when there are too many valid values to display in a drop-down list or list box and the users need to be able to search the values for the one they want. The Look Up dialog window would be similar to the Find dialog window with the addition of a list box to list the choices found.

Andy Cubbon was part of the IBM Dallas Open Systems Center before leaving to set up his own business, User Interface Design Inc. He is a user interface design architect who has been doing user interface designs and reviews for clients for six years. Andy can be reached at (404) 971-8552 or via e-mail at 75357.241@compuserve.com.

REFERENCES/ENDNOTES

IBM VisualAge User's Guide and Reference (IBM Doc. SC34-4490)

IBM Smalltalk Programmer's Reference (IBM Doc. SC34-4493)

1. To be consistent with other GUI tools, VisualAge uses the terms "part" and "script" instead of "class" and "method." I use the Smalltalk terms here.



REXX Diagnostic Commander Makes REXX Debugging Easy

Software developers, administrators, and all OS/2 professionals can efficiently code and test REXX command files in a completely interactive environment. Easy to use, RDC enhances your code writing ability while increasing productivity. *Programs are debugged easily and quickly!*

- Supports Multiple Platforms
- Source Code Display
- Variable Display Service
- Variable Alteration
- Single Step Review
- Dynamic Watch Windows
- Logic Alteration
- Deferred Debug Option

Get the REXX Diagnostic Commander

Only \$45.⁰⁰

Order It Today!

**Call
800-724-7011**



**Suitable
Alternatives**

4655 Old Ironsides Drive, Suite 220
Santa Clara, CA 95054 • 408-727-3142



Circle Reader Service Number 16



This article is excerpted with permission from *The Art of OS/2 Warp Programming*.

By **KATHLEEN PANOV, LARRY SALOMON JR., AND ARTHUR PANOV**

Some GUI Basics: Window Messages and Queues



Kathleen Panov



Arthur Panov

Presentation Manager windows communicate using a queue message processing system. All windows in a Presentation Manager thread share a single message queue for processing messages; however, all message queues are descendants of the Presentation Manager system message queue. This is the reason that one poorly designed Presentation Manager application can freeze up the entire system. The queuing mechanism is an important concept to understand.

Once a window has a message queue, it can communicate with any other window in the entire system. All it needs is the window, or message queue, handle to send the message to.

A window can send or receive messages. Each message is used to signal some sort of event. Each time a mouse is moved, a window is resized, or a menu item is selected, messages are sent to a window. A window procedure operates like a massive sieve, filtering the messages of interest and passing through those messages that are unimportant. It is important to realize that all messages must be processed and replied to, either through your own window procedure or by passing the message to `WinDefWindowProc` or `WinDefDlgProc`. This facility of using events to control the programming flow is known as event-driven programming. This style is common to not only Presentation Manager programming but other GUI programming environments as well.

MESSAGE ORDERING

It is not a good idea to count on messages arriving in your message queue in a certain

order. The purpose of event-driven programming is to be flexible and dynamic and respond only when asked; however, there are obviously times when it is important to understand the flow of messages the system sends to your queue.

The first message you can count on being sent to your client window is the `WM_CREATE` message. At the time this message arrives, the window handle exists but has no size and is not visible. The `WM_CREATE` message can be used to do some application-specific initialization, such as allocating memory for window words. Any queries specific to size or focus, however, should be done after creation. One way to accomplish this is by posting a user-defined message to the client window in the `WM_CREATE` processing. The size and focus messages the system places in the queue are sent messages and will be processed before a posted message. When you process the user-defined message, you will have a client area that has both size and focus, and this information can be used in any initialization that needs to be done.

The standard way to set size or focus is by using the respective APIs directly after the `WinCreateStdWindow` or `WinCreateWindow` call. You can change the size or position of a window in two ways. The first way is to create the client window as not visible and then use the function `WinSetWindowPos` to size and show the window. The second method is to intercept the `WM_ADJUSTWINDOWPOS` message. This message is sent before a window has been sized or moved. This gives the application a chance to override the new size and position with a size and position of its own choosing. If modifications are made, the application

From the *Art of OS/2 Warp Programming* by Kathleen Panov, Larry Solomon Jr., and Arthur Panov. Copyright © 1995 John Wiley & Sons Inc. Reprinted by permission of the publisher John Wiley & Sons Inc. To order a copy of this work call (800) 225-5947.



should return `TRUE` instead of `FALSE`, and the new coordinates are used.

FOCUS MESSAGES

When a window is gaining or losing focus, several messages are sent by the system. It is not advisable to process any of these messages yourself, but it is useful to understand how Presentation Manager handles changing a window's focus.

When a user clicks the mouse on another window, the system first sends a set of messages to the window that is losing the focus. A set of `WM_QUERYFOCUSCHAIN` messages are sent to the frame window and its children to help the system decide which windows will be involved in this focus change operation. Next, a `WM_FOCUSCHANGE` message is sent to both the frame and its children to indicate they are all losing focus. The next message sent is the `WM_SETFOCUS` message. This message indicates the window is either about to lose or about to gain the input focus. In this case, it would be losing input focus. Next, the `WM_SETSELECTION` message is sent. This message is used to unhighlight or highlight any selected items in the window. The client area does not do much with this message, but it is at this time that the titlebar window changes from a highlighted titlebar to an unhighlighted titlebar. The last message sent when a window is losing focus is the `WM_ACTIVATE` message. The message actually takes away the focus from the active window.

When a window is gaining focus, the messages are sent in a similar fashion. First, the system queries the windows with the `WM_QUERYFOCUSCHAIN`. Then, a `WM_FOCUSCHANGE` message is sent to the frame and its children to indicate they are gaining focus. Next, the focus change operations are actually performed, with a `WM_SETFOCUS` being sent first, then the `WM_SETSELECTION`, and lastly, the `WM_ACTIVATE`.

SIZE AND PAINT MESSAGES

An application receives three messages when a window is sized, `WM_CALCVALIDRECTS`, `WM_SIZE`, and `WM_PAINT`. The message `WM_CALCVALIDRECTS` is used to communicate the new window size and coordinates after the sizing operation. The `WM_CALCVALIDRECTS` is used only when `CS_SIZEREDRAW` style is not specified since the whole window will be invalidated when a sizing operation is done on a window with this style.

The next message is the `WM_SIZE` message. This message gives the application a chance to reposition any other window that may be dependent on the newly sized window's position. The last message passed, if the style `CS_SIZEREDRAW` is set, is `WM_PAINT`. If the `WS_SYNC-PAINT` style is set, the message will be sent, otherwise the message will be posted. The system will pass the rectangular coordinates that contain the area to be redrawn as a parameter in the `WM_PAINT` message.

THE LAST MESSAGES A WINDOW RECEIVES

When a `WM_CLOSE` message is posted to a window (when the user selects `CLOSE` from the system menu), a `WM_SYSCOMMAND` message is posted with the `SC_CLOSE` ID. Next, a `WM_QUIT` message is posted to the message queue. This is a very special message because when `WinGetMsg` receives it, the function returns `FALSE`, causing the `WinGetMsg/WinDispatchMsg` loop to terminate. A `WM_SAVEAPPLICATION` message is posted next. This gives the application a chance to prompt the user for any last-

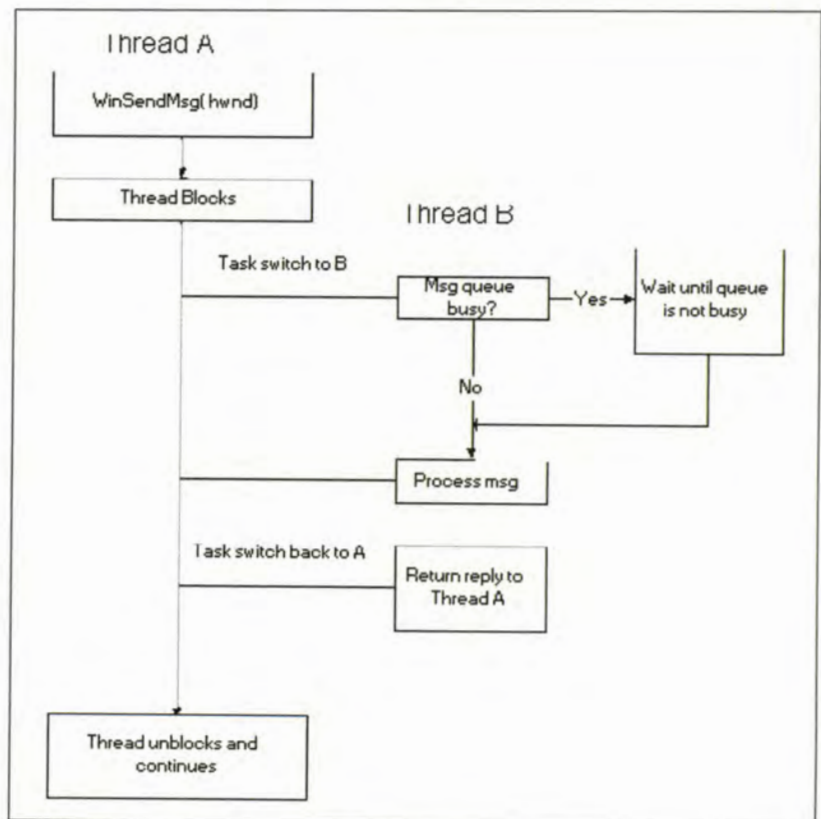


Figure 1. `WinSendMsg` in a multithreaded application.

minute cleanup work, such as saving a file or disconnecting a communication line. When `WinDestroyWindow` is used to destroy the frame window, the system will send the focus change messages to indicate this frame window and all its children will be losing focus. The last message a window will receive is `WM_DESTROY`. This is the place to control any application-specific cleanup. For example, freeing memory should be done in the `WM_DESTROY` processing.

When the user has selected "Shutdown" from the desktop menu or the Warp Launchpad, there is a little change in the messages that arrive in the queue. The system bypasses the `WM_CLOSE` message and sends two messages to each thread that contains a message queue. The first is the

`WM_SAVEAPPLICATION`. The next message issued is the `WM_QUIT` message. An application will usually not process the `WM_QUIT` message; however, in the case when it needs to interrupt or halt system shutdown, it must process `WM_QUIT`. If the application wants to cancel the shutdown, it can call `WinCancelShutdown`. If the application would like to do something else before shutting down, it can perform its closing work and then call `WinDestroyMsgQueue`. After processing, make sure you return `FALSE` implicitly, and do not call `WinDefWindowProc` as the default window procedure does not know how to handle a `WM_QUIT` message.

GOTCHA!

For each thread that contains a message queue, make absolutely sure you

issue a `WinCancelShutdown` soon after the thread is created if you do not want to process the `WM_QUIT`, or else be prepared to process the `WM_QUIT` message and destroy the message queue. A thread with a never-ending message queue can prevent the entire system from shutting down properly. Also, there is no guarantee that a secondary thread will execute all function calls and return before the primary thread (and thus the application) exits. It is up to the developer to make sure all cleanup in secondary threads is complete before the application exits.

SENDING MESSAGES

When a message is sent, it is usually directed to a particular window. For instance, a `WM_CHAR` message indicating a key had been pressed would be sent to the window that was currently active and had the keyboard focus. Messages can be dispatched in two ways. They can be either sent, using `WinSendMessage`, or posted, using `WinPostMessage`. There is a very subtle difference between these two dispatch methods, and this could cause you problems somewhere down the road. When a message is sent, it is not put in a window's message queue; it is processed the next time `WinGetMsg` is called or immediately executed if no message is currently being processed. The thread containing `WinSendMessage` blocks, and control is switched over to the thread containing the receiving message's window procedure.

A message should be sent when it absolutely, positively has to be there right now. A good example of this is passing pointers in messages when there is no guarantee that the pointer will point to something valid when the message is up for processing. `WinSendMessage` should be used in this situation.

GOTCHA!

One little bit of information about `WinSendMessage`: this function will not return until that message has been processed. Yup, that's right. If you send a message from your window procedure to a window procedure that's asleep at the wheel, or even just a little slow to respond, your window procedure will sit there and wait until it gets some response back from the other window procedure. If you send a message to some window for which the system controls the window procedure, you

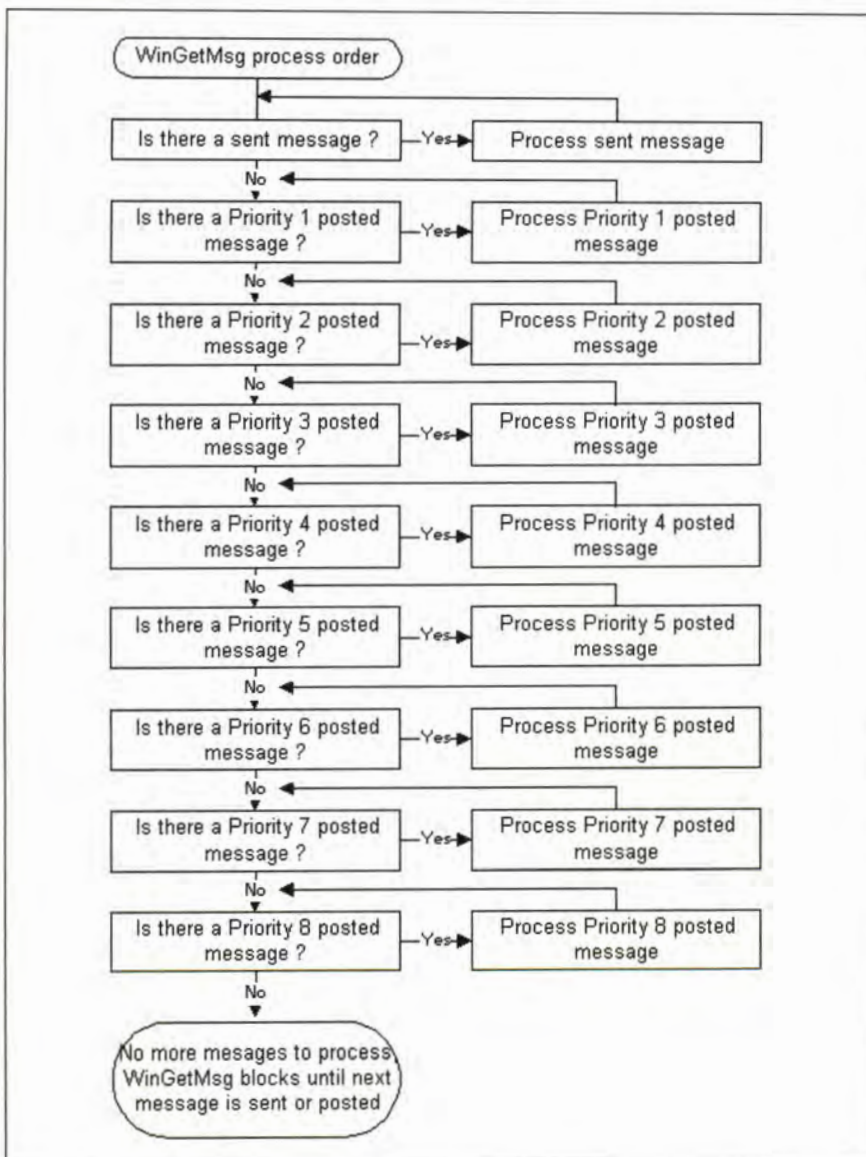


Figure 2. `WinGetMsg` message processing order.



can pretty much guarantee a zippy response. However, be careful when using this function to send messages to either your own window procedure or to some other application's window procedure. `WinPostMsg` is a much safer method of transmitting messages, although the message is placed into the receiving window's message queue. It will be processed when that window gets around to it. `WinPostMsg` should be used when you want to communicate some information and do not care about a reply. `WinSendMsg` should be used when it is imperative that you gain some piece of information and have to respond to it now.

BROADCASTING MESSAGES

A window can communicate one on one with another window directly, or it can broadcast a message to several windows at once. The function:

```
WinBroadcastMsg( HWND hwnd, ULONG uMsg,
MPARAM mp1, MPARAM mp2, ULONG uCmd )
```

can be used to send or post a message to the windows specified in the `uCmd` parameter. This command contains two parts: with whom to communicate and what form of communication to use. These flags are then ORed together. The default communication form is `BMSG_POST`. You can specify `BMSG_POST`, `BMSG_SEND`, or `BMSG_POSTQUEUE`. The `POSTQUEUE` flag will post a message to all threads in the system that have a message queue, and the `hwnd` parameter will be ignored. Only one of these three flags can be specified. The second part of the `uCmd` parameter indicates with whom to communicate. The choices are `BMSG_DESCENDANTS` or `BMSG_FRAMEONLY`. `DESCENDANTS` will communicate with `hwnd` and all of its descendants. `FRAMEONLY` will broadcast a message to all frame windows that are descendants of `hwnd`. To broadcast to all frames in the system use `HWND_DESKTOP` for `hwnd`.

PEEKING INTO THE MESSAGE QUEUE

In many instances, you do not want to retrieve a message from the message queue but instead would rather just peek into the queue and see if a message is waiting. The function:

```
WinPeekMsg( HAB hab, PQMSG pqmsg, HWND
hwnd, ULONG uFirst, ULONG uLast, ULONG
uOptions)
```

inspects the message queue and returns information about the queue. `hwnd` narrows the search to a specific window or its children. The `uFirst` and `uLast` parameters let you narrow the search even further to a numerical range. If both these parameters are null, all messages are included in the search. The `uOptions` flag indicates whether the message is removed from the queue or not. The default is to not

remove the message from the queue. The return from this function indicates whether the search was successful or not. Table 1 shows the several functions that query information from the message queue.

MESSAGE PRIORITIES

When messages are retrieved from the message queue, they are not necessarily retrieved on a "first in, first out"

You're looking at a Rocky Mountain Bighorn Sheep. But no one calls this bighorn sheepish. He's intelligent he's energetic, and he thrives on being at the top. He knows he'll have to butt heads with some powerful challengers, so he is always well prepared for fierce competition.



If you are an OS/2 programmer who has escaped from the sheepish flock, you already know you cannot miss ColoradOS/2. But don't hesitate - space is limited, and you can't afford to wait for another whole year. In fact, that would be very baaaad!

Plan now to
attend the 4th
International
ColoradOS/2
!!!

OS/2 Programming
Summit!

ColoradOS/2

OS/2 software developers have refused to join the "everybody follow me" flock. They won't tolerate having their vision limited by the south end of their closest competitors. Or having everyone's vision limited by a self-appointed leader. Then ending up fleeced.

And every Fall, the best OS/2 programmers in the world gather in the Colorado High Country, for a week of intense OS/2 programming information exchange. Mingling with OS/2 gurus and gifted newcomers, eagerly sharing their knowledge, making lifelong friends and tapping into a worldwide network of people all committed to excellence.

October 15-
October 20, 1995

Keystone Conference
Center
Keystone, Colorado

To register or for information call:
(800) 481-3389 US & Canada
(719) 481-3389 International
(719) 481-8069 FAX

OS/2 is a registered trademark of International Business Machines Corporation.

Circle Reader Service Number 17

Function	Description
WinQueryMsgPos	Returns the pointer position when the last message retrieved from the queue was posted. This is the <code>ptl</code> parameter in the <code>QMSG</code> structure.
WinQueryMsgTime	Returns the time in milliseconds when the last message retrieved from the queue was posted. This is the <code>time</code> parameter in the <code>QMSG</code> structure.
WinQueryQueueInfo	Returns the <code>MQINFO</code> structure. This structure includes the process ID, thread ID, and message count.
WinQueryQueueStatus	Returns information about what types of messages are in the queue.

Table 1. Finding more message queue information.

basis. Instead, messages are retrieved on the basis of priority, similar to threads. Following is a list of messages in the order they will be retrieved:

- Sent messages
- WM_SEM1
- All other posted messages
- Keyboard or mouse messages
- WM_SEM2
- WM_PAINT

- WM_SEM3
- WM_TIMER
- WM_SEM4.

Figure 2 represents a flow chart of how messages are retrieved.

You may be wondering, What are these WM_SEM messages, and what is the WM_TIMER message? Well, on to the next topic... WM_PAINT messages are fairly low on the message priority totem pole.

The default window style causes Presentation Manager to group invalidated regions together and generate one WM_PAINT message. The window style, WS_SYNCPAINT, or the class style, CS_SYNCPAINT, will stop Presentation Manager from behaving in this independent manner. And each time a region is invalidated, Presentation Manager will very obediently call the WM_PAINT processing immediately by sending the WM_PAINT message. The system does not post this message; it jumps to the WM_PAINT processing, and then, when painting is completed, jumps back to the call following the region invalidation.

MESSAGES AND SYNCHRONIZATION OF EVENTS

Often an application wants to know when some event has occurred. One way to do this is by using the WM_SEM1,2,3,4 messages. These messages are totally for your application use. If they are passed to WinDefWindowProc or WinDefDlgProc, it has no effect on the system. For example, suppose you have a

ASIAN OS/2 DEVELOPER SUPPORT



Get help porting OS/2 applications for Japanese, Chinese, or Korean.

Experienced guidance from the leading double byte character set OS/2 developer. Training or consulting on either an hourly or long term basis.



MicroBurst, Inc.

9035 Shady Grove Court
Gaithersburg, MD 20877

(301) 330-2995

Fax: (301) 330-8609

CompuServe: 70334.3616

Internet: 70334.3616 @
COMPUSERVE.COM.

Circle Reader Service Number 18

What does it take to test a client/server system?

Synchronized, multi-user testing
that goes beyond the types of testing possible with stand-alone, GUI test tools.

Unattended, extended use testing
that lets you determine how your app stands up to continued pressure.

Stress, load, and performance testing
to answer key questions about the activity level your system supports.

The Softbridge Automated Test Facility has been testing client/server systems since 1990. To learn how ATF can help you test OS/2, Windows, and NT apps, call 617-576-2257 (or fax 617-864-7747).



The Softbridge
Automated Test Facility

Softbridge, Inc. ▲ 125 CambridgePark Drive ▲ Cambridge MA 02140

Circle Reader Service Number 19

worker thread that has finished processing. That thread could post a WM_SEM2 to the main thread to indicate that the thread has finished its work. WM_SEM1 messages should really be reserved for very important, time-critical events.

A way to keep track of an event that is dependent on some function of time is to use the functions WinStartTimer and WinStopTimer. WinStartTimer starts an alarm clock that is set to go off after some application-defined amount of time, in milliseconds. When the timer goes off, the system sends a WM_TIMER message back to your window procedure.

You might consider using semaphores in a window procedure. DON'T!! Instead, think of using WinRequestMutexSem, WinWaitEventSem, or WinWaitMuxWaitSem. Waiting on a semaphore using the regular DosWait...Sem functions can bring a window procedure to a screeching halt. Even the most well-behaved semaphore synchronization can develop a mind of its own every now and then. The special set of window semaphore functions

were created to provide the same functionality as the DosWait...Sem functions but not to interrupt the flow of your window procedure completely. The system will appear to wait in the message processing that this function is called from, but messages sent to the message queue will be processed synchronously. When the semaphore has been posted or the function times out, the message processing resumes where the WinWait... call was executed. Note that messages that are posted will remain in the message queue until after the WinWait... call has completed.

USER-DEFINED MESSAGES

Presentation Manager also gives you the flexibility to add your own messages to the system. These are called user-defined messages and are numerically represented by the range 0x1000 through 0xBFFF. Some system-defined messages fall into this range: WM_USER+40 through WM_USER+55. This is an area that may change in the future, so it's a good idea to search through the Toolkit header files to see if there are

any new messages defined that fall into this range. Several examples in this book use user-defined messages.

Kathleen Panov is a software consultant specializing in OS/2 software development. A graduate from Texas A&M in 1987 with a BBA in business analysis, she has been involved with OS/2 development since 1988.

Larry Salomon Jr. has been developing OS/2 applications since version 1.1 in 1989. He has written numerous applications, including the Scramble applet that was included in OS/2 versions 2.0-2.11 and the I-Brow, Magnify, and Screen Capture trio that has been distributed on numerous CD-ROMs. He is the President of IQPac Inc., which is responsible for the publishing of EDM/2, the electronic magazine for OS/2 developers, and he is a frequent contributor to the publication.

Arthur Panov is an independent software consultant currently working on the MAC drivers for OS/2 for the PowerPC at IBM Austin. He received a B.S. in Physics from the University of South Florida and has been involved with OS/2 development since 1989.

Point and click your way to fast development of GUI client-server applications.

Do it with Guidelines, a second-generation object-oriented graphical workbench. Guidelines lets you create graphical client/server applications for OS/2 and Windows clients. A built-in prompter lets you simply point and click or drag and drop from the desktop using more than 30 presentation manager controls.

Guidelines utilizes JBA's high-level object-oriented language (JOT) to describe applications. After the application is designed, JOT automatically converts to C++ code, so you don't have to learn its complex syntax and rules. But if you already know C++ and want to use it directly, you can. The conversion takes place regardless of the platform intended.

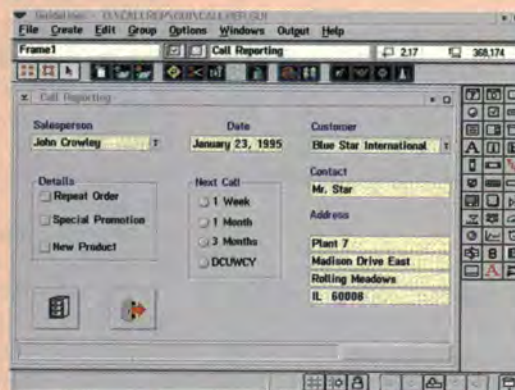
Guidelines saves time and simplifies the creation of graphical, platform-independent applications. Its scalable architecture lets you create anything from stand-

alone PC applications to complex enterprise-wide business solutions. Guidelines supports DB2/2, DB2/400, DB2/6000, DDCS, ODBC, Q+E Lib, and SQL.

Call for a free demo pack: 1-800-JBA-INTL.

In Canada: (905) 940-2442

JBA International
161 Gaither Drive, Ste. 200
Mt. Laurel, NJ 08054





Recent additions to the C++ language now allow developers to reach for a new level of robustness in their software. Of these, exceptions are one of the most powerful. But their optimum use entails a radical new approach to application architecture. **By DEAN RODDEY**

Writing Exceptionally Robust Software



Dean Roddey

As more OS/2-based companies move beyond the stage of OOWT (Object-Oriented Wishful Thinking) and start to implement or port their systems to C++, they need to understand what C++ offers. In this article I will try to shed some light on a particularly powerful C++ tool, exceptions, and how to use them effectively.

PREREQUISITES

This article assumes a working knowledge of C/C++ syntax and at least a basic knowledge of object orientation. You might also want to read "An Exception You Can Handle" (*OS/2 Developer*, January/February 1995) for a discussion of OS/2 system exceptions. It will give you some perspective on the differences between C++ exceptions and OS/2 exceptions. C++ exceptions are similar in concept but quite different in impact and power. C++ exceptions are defined by the language (as opposed to the operating system) and are, therefore, platform independent and object savvy.

ROBUST SOFTWARE

A big problem in software today, as everyone knows all too well, is that it is not terribly robust. We would all like to write manly, hardened, Old Spice-wearing software that never shows its sensitive side, but it seems to elude us. Of course, many people see C++ as anything but robust, being just a more dangerous version of C. I would attribute this attitude to an outmoded antipathy toward all things C. But recent additions to the C++ language, such as RTTI, templates, and exceptions, have made this view even more

questionable. Of these additions, exceptions are among the most powerful tools we have now to create robust software. After recently converting all of my code to an exception-driven architecture, I can personally attest to the power of this tool.

UNDERSTANDING STACKS

To understand exceptions, you must understand the stack concept. Under OS/2, each thread has its own stack. When a thread calls a function or method, a number of things happen on the stack, as shown in Figure 1. Before the call, the compiler will push the arguments that are going to be passed. The order and format generally depend upon the language and the compiler. The compiler then pushes the address of the next instruction after the call and jumps to the function/method entry point. On entry to the function, the compiler will save the stack pointer in the page frame register and then move the stack pointer down far enough to make room on the stack for the automatic variables. This underlying magic happens (with some small variations) regardless of the language or compiler.

So automatic variables are just a blob of space on the stack of the calling thread. This is why multiple threads can call into a function and not step on each other's automatic data toes. And it means that the compiler, on exit from the function, can magically wipe out all of that data just by restoring the saved stack pointer from the page frame register. The data is still there on the stack, it is just ignored now and will be overwritten by the next call. There is no need for the overhead of allocation or deallocation since the



space is already there on the stack, as illustrated in Figure 1. The compiler then issues a RET statement, which causes the CPU to pop the return address off the stack, and returns to the instruction following the original call. At that point, C programs will clean off the pushed parameters. Other languages allow the called function to do that.

That is all well and good, but there is one major shortcoming. Exiting the function does nothing more than move the stack pointer. This is fine unless one of those local variables is a resource handle or a pointer to some dynamically allocated memory. In such cases, the called function is responsible for cleaning up such resources before the pointer or handle goes out of scope. Making sure that all dynamically allocated resources get cleaned up means that either you bend over backwards to have a single function exit (that is, make it very nested) or you make sure that the cleanup is done from every return point. Failure to do this is a prime source of resource leaks, the bane of robust software (made worse by the fact that they don't show up until the program has run for some time).

C++ is much smarter about this situation. It will call the destructor of any objects that go out of scope. For an automatic object, this occurs upon exiting the function or the block in which it was declared. This means that, given a fully object-oriented architecture in which all resources are wrapped in classes, resource leaks of this type can be totally banished with minimal effort. Every object will be given a chance to clean up its resources regardless of where or how the function is exited. This single concept makes C++ orders of magnitude more robust.

For those occasions where you must dynamically allocate automatic objects, you can use the janitor concept to handle resource cleanup. Figure 2 shows the class definition of a janitor. It is incredibly simple and totally inline for good performance, although this example assumes a single-rooted system. The janitor's constructor is given a pointer to a dynamically created object. When the janitor goes out of scope, its destructor will be called, which will delete the object it is controlling. I have to admit I stole the janitor idea from Taligent, though I doubt the people there invented it either.

EXCEPTIONS

Now we can finally talk about C++ exceptions. Exceptions are thrown and caught. They are thrown to report an error and caught to handle the error. C++ exceptions "unwind" the stack. This means that at the point where the throw is done, it is as if an implicit return statement was done. However, this return traces its way back out from each block of code, which includes loops, if blocks, functions, and so forth, looking for a catch block on the stack. In each block, if no catch is found, any automatic objects created at that level are destructed and the search continues. If there is no catch in the function, all of the top-level automatic variables are destructed and the function returns to the caller. The search continues there and so on. If no catch is found, it will eventually hit the run-time library and the process will be terminated. (C++, unlike C, allows you to declare automatic variables within any block of code. The exception destructs each block's objects

The Code

```
ULONG u1FOO(ULONG u1Parm)A
{
    PULONG pu1Ptr = new ULONG[16];
    LONG iTmp;
    ULONG au1Tmp[2];

    ...
}
ULONG u1Res = u1Foo(10);
```

The Stack

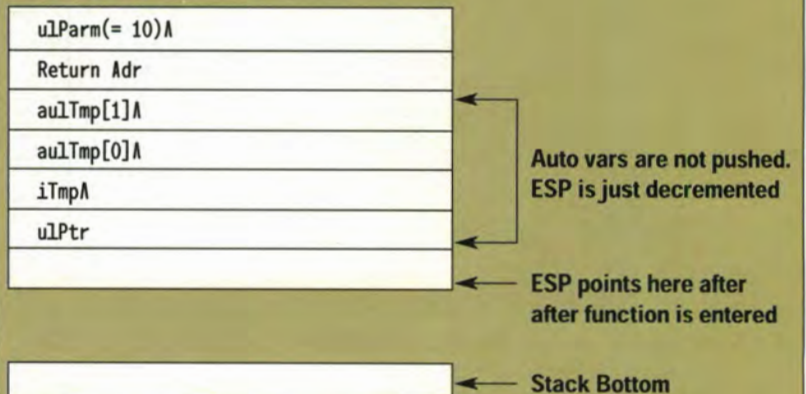


Figure 1. Example of stack usage in a function call.



as it searches upward.)

Here is a list of important rules about C++ exceptions:

- Exceptions are based on the type of the thing thrown. For the most part, only objects are thrown, though fundamental types can also be thrown.
- Any thrown objects must have a valid copy constructor.
- You can have multiple `catch()` blocks after a throw. The first one that matches exactly or can be converted via standard conversion rules or conversion operators will be taken. The others will be ignored. So put the most specific types first, followed by more general ones.
- Any unhandled exception will be handled by the run time and will terminate the application.
- The statement `catch(...)` will catch anything and can be used as the last catch block to handle anything weird coming along. This should not be required if the following strategies are used, though the highest-level code might use one during debugging.
- Inside a `catch()` block, you can just type `throw;` to rethrow the exception after you have handled it, assuming you want it to propagate up the stack. This is preferable to actually rethrowing the caught object explicitly because that might slice the object (Figure 3).
- Don't throw an exception within a destructor (or call anything that

```
class JANITOR
{
public:
    JANITOR(OBJECT* pObjToDestroy) :
        __pObjToDestroy(pObjToDestroy)
    {
    }

    ~JANITOR()
    {
        delete __pObjToDestroy;
    }

private:
    OBJECT* __pObjToDestroy;
};
```

Figure 2. Your very own personal JANITOR will clean up dynamically allocated data on your behalf. It may not be politically correct, but SANITATIONENGINEER was a little long for a type name.

```
VOID UpdateFooBufs()
{
    // These must be zeroed out!!
    PULONG pulBuf1 = 0;
    PULONG pulBuf2 = 0;

    try
    {
        // Allocate the buffers
        pulBuf1 = new ULONG[256];
        pulBuf2 = new ULONG[256];

        // Fill them from a file
        ReadInfo(hFile, pulBuf1);
        ReadInfo(hFile, pulBuf2);

        // Do something to them....

        // And write them back out
        WriteInfo(hFile, pulBuf1);
        WriteInfo(hFile, pulBuf2);
    }

    catch(const HRESULT& errToCatch)
    {
        // If any of these are still 0, that is legal.
        delete pulBuf1;
        delete pulBuf2;

        // And rethrow to caller.
        throw;
    }
    delete pulBuf1;
    delete pulBuf2;
}

INT main()
{
    try
    {
        UpdateFooBufs();
    }

    catch(const HRESULT& errToCatch)
    {
        cout << "Error updating foo info: "
              << errToCatch.strText() << "\n";
        return 1;
    }

    cout << "Updated foo info ok\n";
    return 0;
}
```

Figure 3. An example of a relatively raw use of exceptions to handle the cleanup of nonobject, dynamically allocated resources.



might) unless you catch it within the destructor. I admit this one is tough and is a major shortcoming of the C++ exception architecture. If you don't catch it within the destructor, the object's memory will not be freed.

- There can not be any intervening code between a `try{} block` and its `catch{} block`, although it would sometimes be convenient to do so.

Now let's look at the details of Figure 3. In this example, `main()` calls `UpdateFooBufs()` to allocate some local buffers and read some information from a file into them, update them, and write them back out. `UpdateFooBufs()` uses a `try{} block` around the code to catch any exceptions that might occur while allocating the data or reading the file. Note the lack of error return codes anywhere in the code. If anything goes wrong, the buffers will be freed and the exception rethrown so the caller will know it failed.

Note that the pointers were set to 0 first. In an exception architecture, the `catch{} block` usually does not know where or how the exception occurred. It can only look at the state of the data and act accordingly. Zeroing out the pointers first means that we can call delete on them without worrying if they were ever allocated or not. C++ allows delete on a null pointer for this very reason. Keep in mind that the pointers can not be inside the `try{} block` because they must be visible in the `catch{} block` to be cleaned up.

This example is meant to be very basic and instructive. In reality, I would not allocate any raw buffers. I would declare a couple of my `HEAPBUF` objects, which would clean themselves up as previously discussed, obviating the need for a `catch{} block` in `UpdateFooBufs()`. Figure 4 is a more realistic example. In it, you can see how automatic object cleanup has simplified the picture even more. This code is as clean as it gets and is very robust since there is no elaborate dance to ensure that errors get propagated upward through many layers of code. It will also have better performance since there is only a single `try{} block` instead of two nested tries.

EXCEPTION ARCHITECTURE

So let's go over a number of extremely fundamental architectural differences between an exception-driven system and a conventional one.

Exception-driven systems don't have error return codes, unlike a conventional system, in which each called function or method is followed by a (sometimes elaborate) checking of error returns. When an error is returned in a conventional system, it is usually passed back to the caller, which often passes it back to its caller, and so on. Basically, a lot of work is done manually to achieve the same effect that exceptions provide for free. It is weird at first to look at code that never returns error codes, but you will come to love it. Be sure to make a distinction between errors and states—exceptions are for exceptional occurrences, not expected ones.

Since huge chunks of code can be written as though nothing will ever go wrong, error handling is all done at

the end via a catch block or is just left to the caller to handle. This freedom to think in a straight line is wonderful, and the cleanliness of the code makes the logic clearer and easier to maintain. It also makes that code immune to changes in the error reporting strategy of the code it calls.

All code must be prepared at all times to have its dynamic resources cleaned up in case of an exception. Now you see how the automatic cleanup of resources in an object-oriented system blends seamlessly into the exception architecture. With a little care, all the code you write will successfully clean up its dynamic resources during a regular function exit or when an exception unwinds the stack.

In an exception-driven system, any error can be recovered from, and

```
VOID UpdateFooBufs()
{
    // Create the read/write buffers
    HEAPBUF hbuf1(256);
    HEAPBUF hbuf2(256);

    // Fill them from a file
    ReadInfo(hFile, hbuf1);
    ReadInfo(hFile, hbuf2);

    // Do something to them....

    // And write them back out
    WriteInfo(hFile, hbuf1);
    WriteInfo(hFile, hbuf2);
}

INT main()
{
    // Try to update the foo info
    try
    {
        UpdateFooBufs();
    }

    // Catch any failure, tell user, and exit
    catch(const ERROBJ& errToCatch)
    {
        cout << "Error updating foo info: "
              << errToCatch.strText() << "\n";
        return 1;
    }
    cout << "Updated foo info ok\n";
    return 0;
}
```

Figure 4. An example of a more likely scenario in which objects clean themselves up, so only the calling code must install a `try{} block`.


```

{
    case WM_COMMAND :
        ULONG uId;

        // Validation functions will throw a string on error
        uId = ITM_DLG_NAME;
        ValidateName(uId);

        uId = ITM_DLG_AGE;
        ValidateAge(uId);

        catch(STRING strErr)
        {
            // Show the message
            ShowMsg(hwndDlg, strErr);

            // Put focus back on offending control and return
            WinSetFocus(WinWindowFromId(hwndDlg, uId));
            return 0;
        }
        WinDismissDlg(hwndDlg, 1);
    }
}

```

Figure 5. An example of ad hoc exception handling. Not something to overuse, but very convenient.

this means that there might not be any such thing as a fatal error. Therefore, nothing can be left undone, regardless of the severity of the error. The “just give up” style is convenient but can not be used to create truly robust programs. When porting an existing system, everywhere that the system just fell over it will now have to be modified to clean up after itself—a daunting task in a large system.

THE TROUBLE WITH CONSTRUCTORS

Exceptions cure a long-running C++ problem of constructors that could fail. Unfortunately the syntax of a constructor does not allow the indication of a failure. So one of two equally distasteful strategies has to be used. One is to leave the object partially constructed and require an initialization method to be called by the client code. This is obviously counter to the *raison d’être* of constructors, which is to construct objects. The other is to set a flag member and force the client code to check it afterward—equally distasteful and error prone. Exceptions cure this problem by providing a mechanism to report the failure of an object under construction. (This capability is very important when you want to create persistent container classes that can not be expected to understand special case initialization needs.

AN ACT OF SELF DESTRUCTION

Since an exception unwinds the stack, it is somewhat of an act of self destruction. You need to work with three basic exception strategies according to the circumstances.

Client Handled. Many exceptions, no matter how deeply they occur, are intended for handling by the end client code. They are allowed to propagate up through all general-purpose code until they are caught by some application-specific code layer that is in a position to handle it meaningfully. Client-handled exceptions are the most flexible since the client gets to deal with any errors. However, the client is then responsible for restarting the operation because the stack has been unwound out of the called subsystem, losing any context of the call.

Internally handled. Sometimes a particular subsystem might wish to remain totally fault tolerant regardless of the client code. Such systems would probably catch most or all exceptions that occur within it (or within the general-purpose code it calls to do its work), recovering and/or restarting automatically. The client application never knows unless the subsystem informs it in some way. This scheme suffers a little more in performance (particularly if called repetitively) because each entry to the subsystem must in-

stall a `try{} block`, which involves saving the state of the thread on the stack.

Ad Hoc. For instance, when the user presses Ok on a dialog and I have to validate the input, an ad hoc exception is a clean way to do so. Figure 5 is an example of this type of use. Basically I set a variable to the ID of the control I am about to validate and then call a local validation function. If the validation fails, the function will throw a string object with the explanation of the failure. The catch block will display the error, put the focus back on the bad control, and then return so the dialog is not exited. This code also demonstrates the use of flags to give the exception handler situational awareness (that is, the setting of the control ID into a variable the catch block can see).

ERROR OBJECTS

Finally, you should think about how to make your use of exceptions practical. You cannot require client code to have 10 different catch blocks everywhere they do a catch (which would be required if you allowed the throwing of a lot of different types.) Instead, it is much saner to create an error class and only throw objects of that class or classes derived from it. The basic error class should contain the information that any error handler would need, such as line number, file name, error code, and so on. Once a basic error class is created, particular subsystems can derive from this class to create their own specific error classes that have extra information. Since these new error classes are derived from the base, any code that catches the base class (correctly via a constant reference) will also catch the derived class.

Error objects must uniquely identify the source of the error because the catching of the object can be far away from the source of the throw. So either all error codes across all DLLs must be unique, not a very good architecture for a large system, or the name of the DLL must be embedded into the error object. I opted for the latter so each DLL has the ability to have any error codes desired. You can have a new error class derivative for every DLL, but that does not help the client code that chooses to catch all exceptions via a reference to the base error object class.

TROUBLE REACHING NIRVANA

Exceptions do have some limitations, so everything is not milk and honey. As mentioned earlier, throwing an exception in a destructor is a potential memory leak. Since any called code can potentially throw an exception, any destructor that calls code outside of its very local neighborhood probably should put a dummy try/catch block around it.

Another problem is the "twice the cleanup, twice the cleanup" problem. In those cases where you do allocate raw buffers (such as Figure 3), look at how the same exact cleanup must be done in two places. The exception handler must do it, and the regular exit code must do it. Over time, this could become a maintenance problem if the cleanup is complex or subtly different in the two places. If you don't need access to automatic data, you can create a local function and call it. This would be a very good place for C++ to support nested functions—a la Pascal/Modula2.

Exceptions generally put off the handling of errors until reaching higher-level code. This is a good thing since that code understands the circumstances at hand and can make a better determination of how to recover. But it also means that those few places that handle errors can potentially be responsible for interpreting a large number of errors, as opposed to the error return scheme where the source of the error is obvious and immediate. So the burden of recovery is often heavier on the client code than in a conventional error return architecture.

SUMMARY

It is my opinion that in the big picture, exceptions just can't be beat, and others obviously agree. Taligent, for instance, is a totally exception-driven architecture and makes use of all of the strategies discussed here and much more. Yes, such high-level language facilities cost more in terms of CPU cycles than their more primitive brethren, but memory is cheap compared to bad press and irate users. The larger the system, the greater the payoff in reduced code bulk and greater stability. I personally can not encourage you enough to consider making your next C++ project an exception-driven affair. You might struggle at first to build a formal approach that you feel good with since it is a totally different

world. In the end, however, your product will be the better for it.

Dean Roddey is a senior engineer at Quantitative Medicine Inc. in Annapolis, Md.

QMI is a Marquette Electronics company and makes OS/2-based clinical information systems. Through extensive psychoanalysis, Dean has discovered that he actually has a deep need to be objectified.



REFERENCES/ENDNOTES

"An Exception You Can Handle," *OS/2 Developer*, Vol. 6, No. 5.

"Taligent's Guide To Designing Programs," ISBN 0-201-40888-0.

1. CSet/2++ supports the `_alloca()` function, which will allocate a block of data from the stack. Basically it just bumps down the stack pointer. If the size is known at compile time, this is much faster than allocating from the heap, and it is automatically freed on exit from the function.

2. For a single JANITOR class to work with any type of object, you must have a single rooted class hierarchy. If you have a single class `F000BJ` from which all classes are derived, you know that any object is a derivative of `F000BJ`. So you can use polymorphism to look at any object as a `F000BJ`. If the `F000BJ` destructor is virtual, it will be virtual in all derived classes. So when you delete the object via the `F000BJ` pointer, you know that the correct destructor will be called no matter what type of object you actually have. Otherwise, you must use templates that are beyond the scope of this article.

3. Slicing can occur because thrown objects are thrown by copying them and because the `throw` keyword has special constraints. So let's assume we have a base error class `ERROBJ` from which we derive a `NETERROBJ`. In this example, a call to a net function throws a `NETERROBJ` which is caught by the caller. `FooCatcher` catches the object as a constant reference to an `ERROBJ`. This is appropriate because it allows him to catch any object derived from `ERROBJ`. However, it then rethrows it explicitly, causing a slice. The reason is that the `throw` keyword uses the compile time type of the object. So the `ERROBJ` copy constructor is called to make a copy to throw, but that copy constructor will only copy those members that belong to the `ERROBJ` class. The extra members added by `NETERROBJ` will be lost.

```
VOID NetFoo()
{
    if (something wrong)
        throw NETERROBJ("This is the error", ERR_CODE);
}

VOID FooCatcher()
{
    try
    {
        NetFoo();
    }
    catch(const ERROBJ& errToCatch)
    {
        // And rethrow incorrectly!!!!
        throw errToCatch;
    }
}
```




This article addresses the challenges faced when creating an easily usable and maintainable reservations system for a travel agency. By MARK GAYLER and STEWART MANLEY

Designing a Reservations System Graphical User Interface

Anyone who has used the services of a main street travel agency in the past will know that this particular industry is not exactly perceived as being on the leading edge of user interface design and development. Character-based interfaces are the norm, occasionally making use of color. Many of these systems have not advanced beyond glorified 3270 emulations.

In 1993, we were asked to assist a leading U.S. tour operator to begin the process of moving its reservations systems technology firmly into the 1990s by producing a user interface for a cruise line reservation system. With significant expertise in the more traditional host-based online transaction processing (OLTP) and considerable experience in MVS-hosted CICS systems on IBM mainframes, the company required no help in the mainframe area. For the new project, however, the tour operator wanted to leverage those skills while taking a bold, forward-looking step into the client/server arena. To do this, the company wanted to use a token-ring-based network of IBM PS/2 systems running OS/2 2.1, LAN Server, and Communications Manager as presentation clients to a traditional CICS-hosted transaction server. Our brief was to design and implement a modern, flexible interface that would serve as a flagship reservations product for the company.

Such an interface poses challenges in two areas. First, it should be designed from a usability standpoint to closely fit the working methods of the ultimate user, the sales agent. It's not difficult to create impressive interfaces full of the latest design ideas and controls, but if this does not actually answer

the business problem, it is more of a hindrance than a help. Second, it poses technical challenges to the developer to create a coherent, usable, and maintainable support structure to permit the interface components to be modified, extended, or reused at a later date with a minimum of programming effort. We will address both of these areas in this article.

The interface was designed based on a thorough business analysis of the requirements of telephone-based reservation sales agents, involving staff familiar with the process from an early stage. Such agents receive calls, typically from travel agencies but occasionally from the traveling customers themselves. Agents have to perform a careful balancing act: they must be as courteous, helpful, and flexible as possible while keeping the length of the call to the absolute minimum required to make the sale. Wasted time reduces the number of calls agents can process, adversely affecting tour sales and, thus, revenue. On the other hand, the cruise line for which the interface was being produced was quite firmly targeting the very top end of the marketplace, which created a need for the utmost professionalism and flexibility during the sales process; customers at this end of the market frequently "shop" while on the phone to the agent, changing suite types, tour packages, itineraries, and so on, before the agent is finally able to close the sale.

Based on these requirements, it was possible to draw up a set of guidelines to which the interface should be designed, including the following:

- Logical flow. The interface must guide the

Fast Visual Application Development for OS/2 and DB2



If you're looking for fast and easy application development for OS/2, then take a look at the award-winning Watcom VX•REXX visual development environment. VX•REXX lets you build applications to exploit the graphical user interface, multi-threading, and multi-processing power of OS/2. VX•REXX Client/Server Edition gives you the added power to access DB2 or other database systems, manipulate the

data, and chart the results at lightning speed.

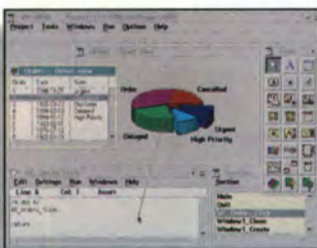
"We like VX•REXX. Using it for development feels like driving a Porsche: it's fast, it's compact, everything's in the right place, and it makes us look good, too."

Peter Coffee, PC WEEK

Designed to Meet Your Needs.

Watcom VX•REXX combines a project management facility, visual designer and an interactive debugger to deliver a highly productive visual development environment. The Client/Server Edition includes additional powerful objects so you can rapidly create rich GUI database applications. You can create OS/2 client applications which connect to DB2/2 or DB2/6000. Use IBM's DRDA support on OS/2 to access DB2 for MVS, DB2/400 for AS/400, and DB2/VSE and VM (SQL/DS) for VM and VSE. Also supported are Watcom SQL and ODBC-enabled databases.

"Overall, this edition of VX•REXX for OS/2 is an outstanding visual client/server development platform." Nicholas Petreley, InfoWorld



Point. Click. And Presto!

To create an application you draw user interface objects, customize their properties using standard OS/2 notebooks, and define their event code using powerful drag-and-drop programming. To add database access just draw a query object, visually design a SQL query, press OK and presto— your window is automatically populated with objects that are bound to your query to display, update and search your data.

"Drag-and-drop nirvana." Nicholas Petreley, InfoWorld

Give Your Data a Whole New Image.

Energize your applications by displaying your data in a 3D chart. The Client/Server Edition gives you more than a dozen chart types to choose from, along with over 150 display options. You also get complete support for run-time events so you can bring new drama to your data by making your chart interactive.

"VX•REXX is a must buy." Jacques Surveyer, ComputerWorld

Standard or Client/Server Edition— Which one is for you?

To start creating powerful OS/2 GUI applications right away, order your copy of Watcom VX•REXX Standard Edition for just...\$99*

Or, to start creating rich client/server database applications, order Watcom VX•REXX Client/Server Edition for just...\$299*

- Over 2 dozen objects, including CUA'91 containers, notebooks, pop-up menus and more
- Integration and control of existing applications through DDE, keystrokes or REXX API's
- Easy to learn event-driven programming model with complete on-line documentation
- Support for professional multi-threaded, multi-windowed and drag-and-drop enabled applications
- Code reusability through section and file sharing
- Graphically create CUA'91 Presentation Manager objects, quickly customize their properties, and easily attach REXX procedures
- Package your application as an EXE or PM macro for royalty-free distribution



1-800-265-4555

Watcom
A Powersoft Company

Watcom International 415 Phillip Street, Waterloo, Ontario, Canada N2L 3X2 Tel. (519) 886-3700 Fax (519) 747-4971 *Prices and specifications are subject to change without notice. Price does not include freight and taxes where applicable. Prices quoted in US dollars.
† ODBC drivers are available from INTERSOLV, Inc. Watcom, the Lightning Device, and VX•REXX are trademarks of Watcom International Corporation Other trademarks are properties of their respective owners. ©Copyright 1994 Watcom International Corporation.

user through the steps required to perform the sales process, from accepting basic customer information to creating the package and closing the sale. It should not allow the user to venture into parts of the process that are not yet relevant since the appropriate data have not been collected.

- **Flexibility.** While providing a structured framework for the sales process, the necessity to be able to handle "shoppers," who frequently change their minds about tour components, requires that the interface allows the user to be able to quickly navigate back and forth to areas of interest at will.
- **Nonintrusiveness.** Informational messages that pop up in the middle of the screen and require confirmation by keystroke or mouse click to dismiss interrupt the user and unnecessarily lengthen calls. All informational messages should, therefore, be placed on a status bar.
- **User levels.** The travel reservation industry makes use of a wide range of short code letter combinations to represent everything from airport and port names to package components, suite types, and dietary requirements. The interface must be able to accommodate the experienced users, who will wish to type in these codes directly, as well as less-experienced staff, who may wish to select items from a list

when they do not yet know the code. In this way, the novice users learn the codes by association as they work.

Based on this analysis, it was apparent that a notebook paradigm most conveniently fitted the requirements for both a logical flow and navigational flexibility. The notebook, oriented with the tabs uppermost in a Rolodex fashion, presents the user with an instantly recognizable and understandable way in which to work. Separate notebooks were allocated to each of the main application modules: the reservations flow itself, along with modules for back office staff to maintain agency, inventory, pricing, and group allocation details.

Only one of the notebooks is visible at any one time, so the frame controls for the notebook's owner window were hidden, making the notebook appear to be floating in the center of the application's main window (Figure 1), reducing screen clutter. Navigation between notebooks is achieved using the action bar or tool buttons. The main window has, in addition to standard frame controls, a combined status bar and toolbar, which displays information relating to the current caller and provides a shortcut to actions in the booking process. A message bar is also present at the foot of the main windows to display the nonintrusive messaging.

The requirement to cater to both

novice and power users was handled by creating a control pair, which became known as a "code/description lookup." It consists of an entry field and combo box (CBS_DROPDOWNLIST style) pair aligned horizontally with the entry field on the left. Examples of these can be seen in the upper part of the panel in Figure 1. Internally, the controls are linked by a simple, sorted lookup table structure, which matches a code (usually two or three alpha characters) to a more descriptive string. For example, "LHR" might be matched to "London Heathrow Airport." If a code is entered into the entry field on the left, the lookup is performed automatically on each change to the entry field. As soon as a match is found, the corresponding descriptive string is highlighted in the combo box. Alternatively, a less-experienced user can select the appropriate descriptive string in the combo box, and the corresponding code will be inserted into the entry field by the lookup process.

GRAPHICAL CABIN SELECTION

Graphical assistance for the process of selecting a cabin was also desirable. We adopted an approach using a value set to represent the ship's deck, with color coding used to indicate the status or class of cabin in use (Figure 2). The original intention was to use a bitmap symbol to denote this, but the processing time required to repaint a large value set containing bitmaps was prohibitive. From this panel, the agent could display bitmaps of both the overall deck layout and the design of an individual cabin, scanned from the ship's documentation (Figures 3 and 4). This is of relatively limited use to the telephone sales agent, who must relay the description to the caller, although it does permit the agent to more easily answer questions about the relative placement of the suite with respect to public areas on the vessel as well as the layout of the cabin. The images will come into their own, however, as such systems are deployed to travel agencies where they can be shown directly to the potential traveler. Additionally, they may be expanded in the future to be able to display digital video clips and sound.

All visual elements within the application were created using Prominare Designer. Once code had been

The screenshot shows a software interface for travel reservations. At the top, there's a header bar with fields for Country (US), Call origin (SC), Caller (STEWART), and Agency (CERTIFIED VACATIONS). Below this is a tabbed interface with tabs for Caller, Search, Available, Suite, Air, Hotel, Options, Price, Payment, Guests, and Summary. The 'Available' tab is selected, showing a 'FIT Booking - New' form. This form includes fields for Cruise line (SC SILVERSEA), Ship (SC SILVER CLOUD), Sail dates (06-30-94 thru 09-28-94), Area, Cruise length, Theme, Group number, Cruise number, Suite category, Suite type, Suite number, Suite name, and a checkbox for Handicapped. Below the form is an 'Availability' table with columns for Date, Days, Line, Ship, Area, Availability, Cruise only prices, and Number. The table lists several options for different dates and ships, with prices ranging from 20.00 to 100000.00.

Date	Days	Line	Ship	Area	Availability	Cruise only prices	Number
07-01-94	10	SC	SC	DUCHDAN	LIMITED	20.00 to 100000.00	1413
07-01-94	20	SC	SC	MED	AVAIL	2300.00 to 2400.00	1413A
07-01-94	20	SC	SC	MED	AVAIL	2300.00 to 2400.00	141314
07-11-94	10	SC	SC	MED	LIMITED	1000.00 to 100000.00	1414
07-11-94	3	SC	SC	MED	AVAIL	2300.00 to 10000.00	1414C
09-21-94	11	SC	SC	CANCOLUS	LIMITED	4395.00 to 30990.00	1421

Figure 1. Reservations notebook interface.

generated by the tool, custom application code was added by hand. Prominare was chosen not only for its excellent resource editing capabilities, but also because its intelligent code generation engine does not overwrite custom application code when the interface code is regenerated after modifications to the panels. Prominare also provides a useful range of custom interface controls, such as the recessed three-dimensional text field, as well as the ability to create further controls. The three-dimensional group box frame seen in the figures is an example of such a control.

DEVELOPMENT

The application code fell essentially into two main core code engines: the support for the notebooks and their navigation; and the mechanism controlling the way data was both retrieved from host transaction server programs and displayed in the panels and read from the panels and returned to the host. This latter engine became known generically as the parsing engine since its most fundamental task was the parsing of data into and out of transaction buffers transmitted to and from the host server programs.

NOTEBOOK MANAGEMENT

Of these two engines, the notebook management was the least complex. Since speed of navigation through the application was a prime business requirement, it was clear that adopting the standard approach of creating each notebook from scratch each time it was surfaced (inserting all its pages and initializing any local data) would cause an unacceptable performance degradation. Therefore, an early decision was made to preload all notebooks and their pages during application startup, also initializing any static data such as lookup tables at this time. Notebooks could then be surfaced or hidden quickly using `WinShowWindow`.

The notebook engine was driven by two data structures. The `NOTEBOOKDATA` structure manages all notebook-level information, such as owner and notebook window handles, number of pages, and current page number. An array of `PAGEDATA` structures maintains information about each page in the notebook, such as its page number, window handle, dialog procedure address, and tab and status text. It also

tracks the ID of the first field on each page so the focus can be set appropriately as pages were turned. This page instance data was stored with the notebook page using the `BKM_SETPAGEDATA` message. Notebook management functions, responsible for creating and populating notebooks and their pages, were created.

Cosmetic features were also implemented. For example, the width of major tabs was calculated at run time based on the width of the notebook and the number of pages it contained. This allowed the tabs to neatly fill the complete notebook width to improve visual appearance. Additionally, the standard left and right arrow buttons, normally used to navigate forward and backward in the notebook, were deleted and customized buttons added to the bottom of each page. The button used to move to the next page was given default focus so that hitting Return moved the agent to the next page.

In addition, each page has associated with it a status, including values such as `COMPLETE`, `INCOMPLETE`, and `DISABLED`. This status determines the way in which the notebook is navigated by the user. For example, during the creation of a new booking, it is clearly not possible to select a suite until such time as a cruise has been chosen. Each status is associated with a color, which is used to draw the text on the page's tab. The status flag and a set of page management functions manage this process.

DATA PARSING AND DISPLAY

It became clear early in the design process that most of the application's operation was involved in accepting data entered into panels by the user, formatting these data into buffers to be transmitted to the transaction server programs on the host, and, once a response had been received, reversing this operation to display the results. Transaction data buffers consisted of the field values in ASCII text, concatenated to form a single text string. Clearly, coding this parsing process by hand for each individual page was unacceptable from several standpoints: it would entail vast code duplication, greatly increase the complexity of each module, increase the likelihood of errors, and create the potential for maintenance problems. A common engine, therefore, was designed to drive this process.

This engine comprises the bulk of the application's core code. A complex data structure drives the engine. This data structure describes an individual field in the transaction buffer, holding information such as field name, type, format and length, the notebook, page, control ID and type with which it is associated, any field-subclassing options, and, finally, a pointer to the field's actual storage within the application. The structures are then aggregated into arrays, each array describing the complete set of data associated with a single transaction server call, initialized as static data within the appropriate source module. Extensions to the

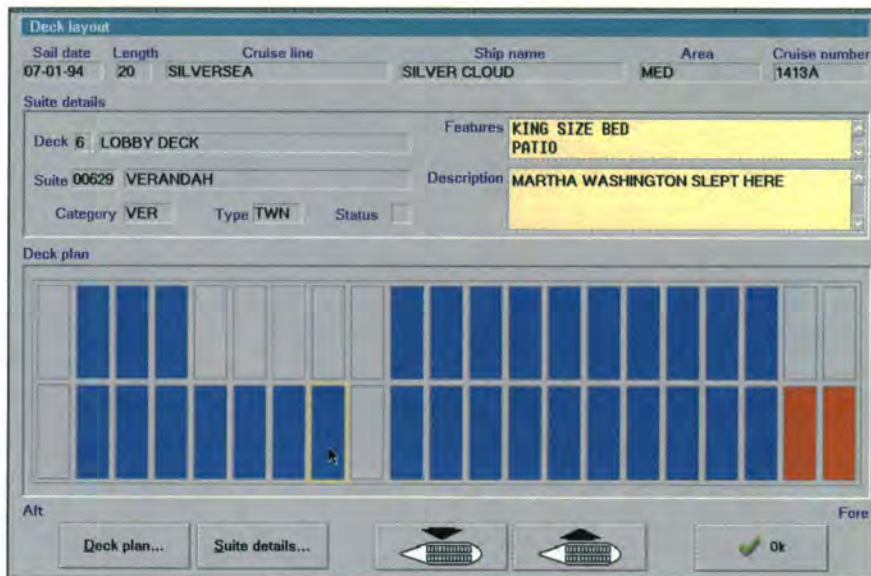


Figure 2. Graphical deck layout using value set.

field type and control type allow the mechanism to manage simple lists and complex repeating data structures (OCCURS clauses, for the COBOL-conversant), usually mapping these to simple or multicolumn list boxes on the notebook page (such as the Availability list box in Figure 1).

Two main layers make up the complete parsing engine: the panel-related code and the data buffer handling. Each layer has two principal entry points: `ReadPanel` and `WritePanel` for the user-interface-related code and `FormatBuffer` and `ParseBuffer` for the transaction buffer management. The principal argument to all of these func-

tion calls is the start element of an array of data structures as previously described. Thus, the developer creating the code to handle a new panel needs to be aware only of these four entry points and able to describe the transaction in terms of the driving data structure, rather than the complete parsing process. Other helper functions, also driven by these descriptive data structures, are also available to the developer. These include `ConfigurePanel`, which manages the process of subclassing data fields appropriately (for example, to restrict input to numeric characters only), `VerifyPanel` for performing data verifi-

cation checks, and `ClearPanel` for quickly deleting the contents of all fields in preparation for new work. Within the parsing engine itself, individual functions for each field type and control type are called to manage each field in the buffer and control on the panel.

As would be expected for any such application, the communications with the host transaction server programs is managed by a secondary thread with an associated object window. Outgoing and incoming calls are managed asynchronously, so the call is dispatched and control returned to the user interface while the secondary thread waits for the response. This response is then returned along with a user-defined message to the appropriate window. This allows status indicator and real-time clock to be continuously updated while the call is in progress.

VISUAL AIDS

Wherever possible within the application, bitmaps are used on push buttons in place of text. Standard images, such as the binoculars for the Search button, were adopted throughout the reservation system. This allows the user to quickly navigate the system.

Early in the development cycle, it became apparent that users were experiencing difficulty in discerning the field that currently had input focus, particularly in complex panels with many entry fields, due to the small size of the caret. This was ameliorated by adding handling such that the control gaining the input focus changed its background color from the default to cyan and then restored the default color when the focus was lost.

IMPORTANCE OF PROTOTYPING

This project provided a perfect illustration of the importance of accurate prototyping of user interface objects and paradigms—or, more accurately, the assessment by the developer of the feasibility of such an interface design.

The interface provided a textual equivalent of the suite allocation system, the graphical version of which used a value set as previously described. Use of drag-and-drop had been vetoed at an early stage for a number of reasons, so the onus was on the interface designer to create a method of linking a set of passengers with one or more suites.

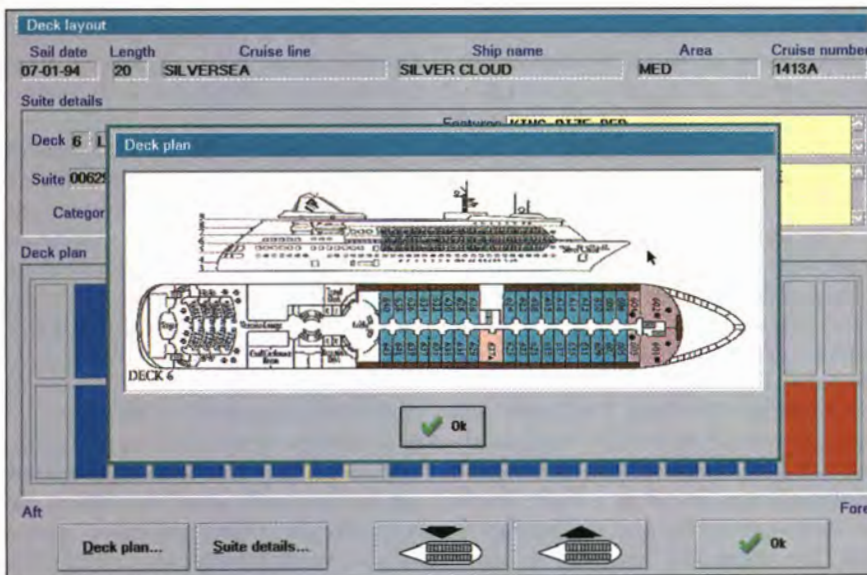


Figure 3. Deck plan image.



Figure 4. Suite plan image.

The result was a pair of list boxes, one of which listed the passengers and the other the selected suites, one above the other. Between the two was a graphical "link" button. The idea was a simple one: the agent would highlight the set of passengers for a suite, highlight the suite, and select the link button. To display the linkage between passengers and suite, selecting either the suite or any one of its occupants would automatically highlight both suite and occupants. Moving a passenger from one suite to another could be achieved by removing his linkage to one suite and creating a linkage to another.

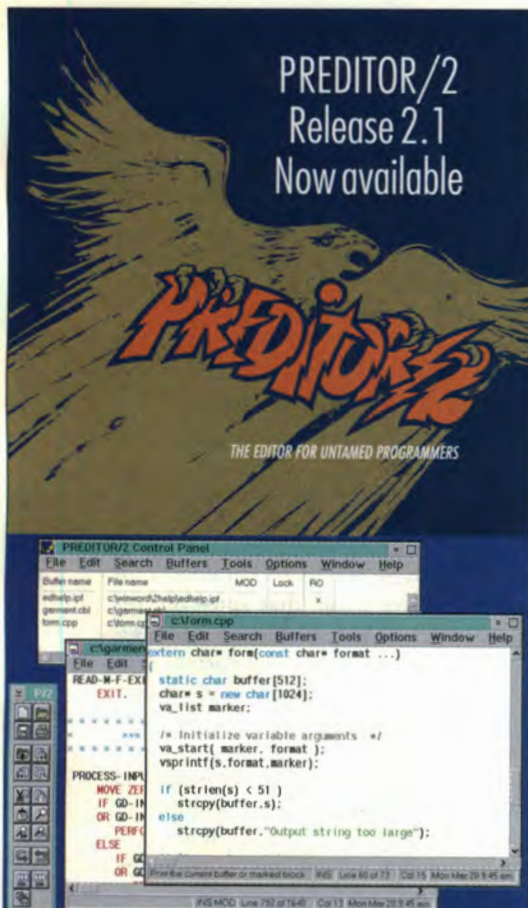
However, when the time came to code this apparently simple and elegant interface, we discovered a number of

problems. How was the code to know if the user was selecting for the purposes of displaying the link or editing it? In other words, the action of the interface was inherently modal, so radio buttons had to be added at a late stage to enable the user to switch between display and update actions. Furthermore, the actual code required to manage this interface became clumsy and difficult to maintain, though the interface itself was usable. Had the developer taken a careful look at the original interface design, from the point of view of how it would be coded, this situation would have been avoided.

Mark Gayler is a cofounder and technical director of Microtransfer Ltd. and has consider-

able experience in the practical application of distributed systems in business environments. Mark was born in the U.K. and was educated at the British Columbia Institute of Technology, Vancouver, Canada. Mark can be reached at Microtransfer Ltd., Park Farm Technology Centre, Kirtlington, Oxon OX53JQ, U.K. or via e-mail at mgayler@microtransfer.co.uk.

Stewart Manley is chief systems architect at Microtransfer and is a specialist in the design and development of OS/2 cooperative systems. Stewart has an MSc in Cardiovascular Studies from the University of Leeds, which comes in handy when SOM programming. Stewart can be reached at Microtransfer Ltd., Park Farm Technology Centre, Kirtlington, Oxon OX53JQ, U.K. or via e-mail at smanley@microtransfer.co.uk.



For untamed OS/2 programmers

Your source code has never fallen into the grips of a more powerful editor. PREDITOR/2 is for developers who don't have the time to deal with rigid programming environments, who must change facilities to fit their exact requirements and who need to dramatically extend editing functions to attain peak productivity. With this editor, you'll work with the speed and strength of a true predator.

HIGHLIGHTS:

- Color syntax highlighting
- Single, multiple and detached window modes
- BRIEF, VI, EMACS, ISPF, CUA emulation
- Hypertext C/C++ browsing
- Built-in compiler and WorkFrame/2 support
- Extensive customization facilities
- Powerful C-like extension language
- Unlimited Undo and Redo
- Powerful search and replace functions
- No limits on file sizes, open files or line lengths

Limited time offer! \$149.00
Order now through Indelible Blue
1-800-776-8284
MSRP \$249.00

PREDITOR/2 is a trademark of Compuware Corporation. All other company or product names are trademarks of their respective owners.
 © 1995 Compuware Corporation.



Circle Reader Service Number 22



Library vendors will usually provide 32-bit versions of their libraries, but occasionally it is not cost-effective to do so. This article discusses how we may still use 16-bit libraries in 32-bit applications.

By John Calcote

Thunking: Using 16-Bit Libraries in OS/2 2.0



John Calcote

Recently I have had the opportunity to adapt several 16-bit OS/2 libraries to a form usable with the newer generation of 32-bit compilers for OS/2 2.0. Though the ideas presented in this article are based on concepts founded by IBM's C Set/2 compiler for OS/2 2.0, they apply equally well to other 32-bit OS/2 compilers on the market.

PROTECTED VS. REAL MODE

Like Intel's real mode, protected mode uses a segmented architecture to describe memory models available to application programmers. Unlike its real mode counterpart, protected mode uses segments the way segments were designed to be used. While real mode uses segments as a sort of Band-Aid to allow the application programmer to access 1 megabyte of memory with only 16-bit registers, protected mode uses segments for virtual memory, for multiple independent application address spaces, and to allow the development of more powerful system tools using simpler memory management techniques.

When OS/2 2.0 was designed, it was decided that a protected mode flat memory model would be used. Since protected mode on the 80386 processor supports 32-bit offsets, all 4 gigabytes of address space can be accessed using only a 32-bit offset—a near32 pointer.

MIXED MODEL PROGRAMMING

To maintain the large investment in 16-bit code that had already been established by OS/2 application writers, IBM gave OS/2 2.0 the ability to run 16-bit applications. In so doing, it created the possibility that someone

would want to call 16-bit libraries from a 32-bit application. It is even possible (although unlikely) that someone would develop a 16-bit application that needed to make calls to a 32-bit library.

This presents a whole range of problems for the operating system developer and the application programmer. However, IBM developed a method in OS/2 2.0 for doing just this sort of addressing mode conversion. In OS/2 terminology, such a conversion is called a thunk. The origin of the word is a little vague, but it stems from compiler technology as a form of parameter passing. The connection between this use of the term and our use is a bit fuzzy, however.

In *The OS/2 Programmer's Reference, Vol. I* (commonly called *The IBM Red Books*), IBM outlines an overview of how these thunk layers should work for the memory management scheme incorporated by OS/2 2.0. More detail on application programming using thunks can be found in volume IV of the same reference. (Note that since the release of OS/2 2.1, IBM has reworked this series into a slightly less rigorous format and has had the series published by QUE Books. They can be found in the computer section of larger book stores in softcover format.)

Three main issues should be considered when writing 32-bit applications that call functions in 16-bit modules: the difference in parameter sizes, the 64K boundary problem, and the difference in call models.

Parameter Size. When an application written in a high-level language makes calls to functions, the most widely used method of passing parameters is by a call frame on the stack. What this means is that parameters

A Special Report from the publishers of OS/2 Developer!

Quantities are limited.
Order now!
For fastest service,
fax your order to us
at 415-905-4967.



Smalltalk



☒ **YES!** Please send me _____ copies of Smalltalk!

Each copy is only \$9.95 in the US or \$12.95 in Canada.

Please add \$2.50 for shipping and handling.

- ☐ My check is enclosed
- ☐ Charge my credit card: ☐ Visa ☐ MasterCard ☐ American Express

Name

Signature

Date

Send them to:

Name

Address

City

State

Postal Code

Mail: Smalltalk
P. O. Box 7046
San Francisco
CA 94120

Phone: 1-800-444-4881

Fax: (415) 905-4967


```

; char dest[100];           ; 32-bit C source code
; char *source = "John";
; strcpy(dest, source);

PUSH    OFFSET source      ; param 2
PUSH    OFFSET dest       ; param 1
CALL    _strcpy           ; function call
ADD     ESP, 8             ; clear params from stack

...

PROC    _strcpy
PUSH    EBP               ; save previous frame pointer
MOV     EBP, ESP          ; make current frame pointer
CLD                     ; clear direction flag
MOV     ESI, [EBP+8]      ; access 'source' as param 2
MOV     EDI, [EBP+4]      ; access 'dest' as param 1

...                       ; actual code to perform copy

MOV     EAX, [EBP+8]      ; return pointer to 'dest'
MOV     ESP, EBP          ; clear any local variables
POP     EBP              ; restore previous stack frame
RET                     ; return to caller
ENDP    _strcpy

```

Figure 1. Sample of assembly code that makes a procedure call to `strcpy`.

USING BORLAND TOOLS

In its current implementation, Borland's OS/2 2.0 compiler supports only the `_Far16` keyword. Its syntax is a bit different in that it is entirely lowercase, `_far16`. Furthermore, Borland's version of this keyword should be used to modify all pointers in return values and in parameters, as well. For instance, while a thunked prototype in C Set/2 might look like this:

```
char * _FAR16 strcmp(char * ptr);
```

the same prototype would be formatted like this in Borland C:

```
char _far16 * _far16 strcmp(char _far16 * ptr);
```

Since the `#pragma seg 16()` directive is also missing, it is possible to pass a pointer to a data structure that crosses a 64K boundary to a 16-bit function. However, it generally requires a fairly large amount of static data to reach this point. If the program is fairly small, the chances of global data buffers landing on a boundary are fairly remote. Still, this is unpredictable and, therefore, dangerous.

Borland suggests not calling 16-bit functions, which require pointers to pointers or using pointers to structures containing pointers as parameters, from programs compiled in version 1.0 of the Borland C++ for OS/2 compiler (Tech Bulletin 1632 — CIS forum BCPDOS/lib2 TI1632.ZIP — Using Sybase SQL libraries and DLLs in Borland C++ for OS/2).

This partial support is not entirely critical, as the situation of a structure containing a pointer can often be worked around.

are pushed onto the stack just before the procedure call is made and then directly addressed on the stack by the called procedure. The source code shown in Figure 1 shows a sample of assembly code that makes a procedure call to the standard C library function, `strcpy`. This function takes two 32-bit character pointers as parameters and returns a 32-bit pointer. This listing also shows the beginning and end of the procedure itself.

A calling convention is a standard method by which procedure calls are made in a high-level language. This includes passing parameters, creating local variables, making the call, and returning return values to the caller. Several high-level language calling convention standards defined by standards committees like ANSI are in existence today. Since this article is mainly concerned with OS/2 2.0, we will discuss the standard C calling convention, denoted by the keyword, `_cdecl`.

In the C calling convention, parameters are pushed onto the stack from left to right, and then the procedure call is made. If there is a return value, it is passed by the procedure in the EAX register. It is the caller's responsibility to clear the parameters from the stack after the procedure returns control to the caller.

The prolog code you see at the beginning of `_strcpy` sets up the call frame on the stack so parameters and local variables may be accessed by a stationary frame pointer. Essentially, the call frame is a structure on the stack whose members are referenced indirectly by the base pointer, EBP. EBP is always used in this manner, so each procedure must assume when it is entered that EBP currently points to a previous call frame belonging to the calling procedure.

Given this fact, the first thing to be done is to save the old call frame pointer on the stack by pushing EBP. Next, the current stack pointer, ESP, is copied into EBP to create the call frame base pointer. EBP is used for this purpose in the Intel architecture because it implicitly references the segment indicated by the SS register rather than that of the DS register. Thus, it is generally not used for any other purpose in a program.

Since parameters are passed on the stack, and the `PUSH` command will only place word-sized objects on the

stack, each parameter uses at least the number of bytes of stack space defined by the machine's basic word size. The problem lies in the fact that OS/2 2.0's basic word size is 4 bytes, or a **dword**. OS/2 1.0, on the other hand, uses a 16-bit architecture, so its basic word size is only 2 bytes. In OS/2 2.0, if the calling procedure pushes an unsigned character (1 byte) on the stack, a full **dword** is actually pushed. If a 16-bit function is called with such a parameter, it will expect two bytes on the stack and will address the call frame erroneously.

A thunk must copy the parameters from the 32-bit stack segment to the 16-bit stack segment, properly realigning the parameters so they are addressable by the 16-bit function.

One more issue to be discussed is that of basic integer size. In a 32-bit compiler, an **int** in the C programming language is 32 bits, or 4 bytes, wide. However, in a 16-bit environment, an **int** is only 16 bits, or 2 bytes, wide. The problem is that if an **int** is passed as a parameter from a 32-bit function to a 16-bit function, the compiler will generate code to pass 4 bytes based on the size of the parameter (not the stack size). The 16-bit function will expect to find a 2-byte quantity—its definition of an **int**—and will access the stack incorrectly. Furthermore, since OS/2 2.0 sees an **int** (4-byte object) being passed, it will not adjust the size of the data object because it is not aware of the differences in the sizes of **ints** on the different platforms.

The fix for this is not to use types whose sizes are environment dependent. Instead, use **short** or **long**, as the sizes of these types are not determined by basic word size. If the library writer used **int** in the 16-bit library, we must rewrite the prototype in the author's header file to use a **short** instead.

The 64K Boundary. To create an environment in which both 16- and 32-bit code can reside simultaneously, IBM redesigned memory management under OS/2 2.0. Since the same memory must be addressable by both types of code, some kind of conversion scheme from one type of address to the other must exist. This scheme should be efficient and reliable. It should also be algorithmic in nature so a high-level language compiler can reliably generate a deterministic two-way conversion for any address in the application's address space.

In OS/2 2.0, four types of memory are available to an application. The first type is shared with the system call interface. The next is shareable memory, or memory that may be shared by two or more processes. The third is reserved for 32-bit code and data only. The last type lies in an area called the compatibility region and may be

shared by both 32-bit and 16-bit code.

In protected mode, segment registers no longer point directly to a physical location in the machine's address space. (Note that we will disregard paging in this article because its operation is completely transparent to the concepts we are discussing.) Instead, segment register values (now called se-

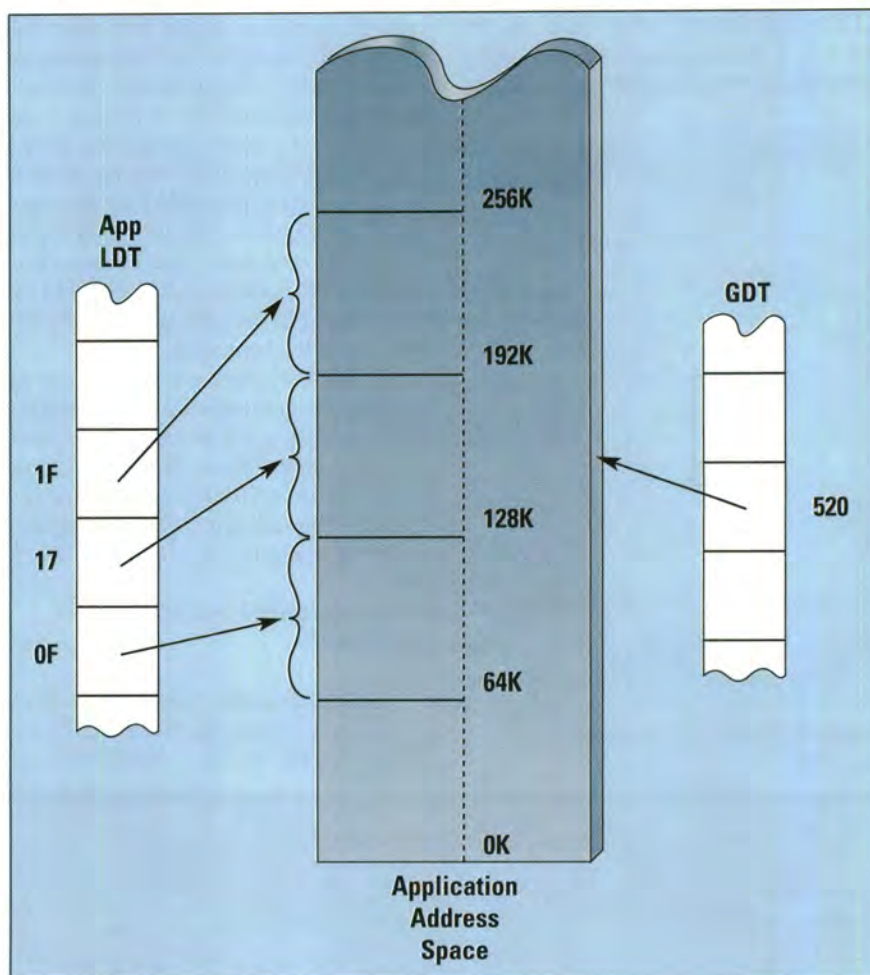


Figure 2. OS/2 tiled memory scheme.

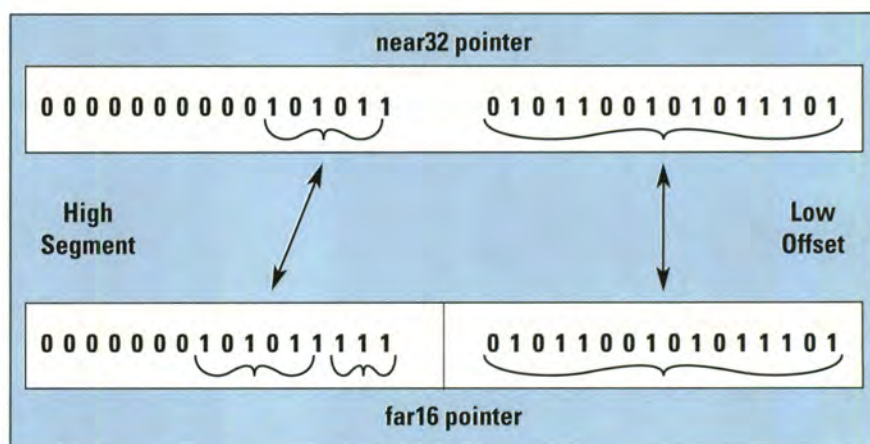


Figure 3. Thunk conversion equations.

lectors) index tables of 8-byte data structures called descriptors. These tables of descriptors are the Global Descriptor Table (GDT) and the Local Descriptor Table (LDT). Each table entry describes a segment—its location, size, type, and protection options.

When a selector register is loaded by a program instruction, the CPU uses the value stored as an index into the GDT or LDT. When the table is accessed, the entire 8-byte descriptor is cached in registers internal to the CPU for quick access.

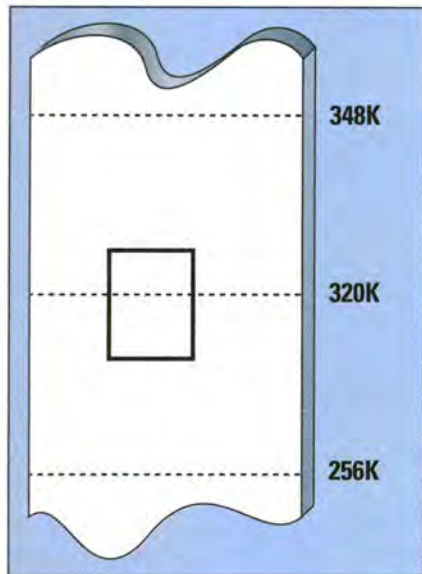


Figure 4. Data structure crossing a 64K boundary.

Descriptor tables must be contained completely within a 64K segment because selector registers are only 16 bits wide. This being the case, only 8,192 8-byte descriptors will fit into a descriptor table.

This is important because the algorithm used to convert a memory address from near32 format to far16 format (and back) is based on mapping specific memory ranges into each descriptor in the LDT. This algorithm is called the Compatibility Region Mapping Algorithm (CRMA). The CRMA uses a form of memory mapping called tiling. What this means can more easily be described by the diagram in Figure 2 than by words. Essentially, memory is laid out in contiguous 64K regions, each accessible by a descriptor in the LDT obtained mathematically by a mapping formula.

In this scheme, selector mapping is a simple mathematical process performed with quick shifts, rotates, and ORs. By using tiling, the addressing schemes of far16 and near32 are related by the following C language formatted equations.

$$\text{near32} = \text{SEL}(\text{far16}) \gg 3 \ll 16 + \text{OFFSET}(\text{far16})$$

Shifting right three places and then left 16 places has the effect of zeroing out the bottom three bits and

then shifting left 13 places. This is important because these bottom three bits are not technically part of the selector index value. Rather, they are the table indicator (TI) and requesting privilege level (RPL) bits. These three bits make no sense in a near pointer.

$$\text{far16} = \text{MAKEP}(\text{HIGH}(\text{near32}) \ll 3 + 7, \text{LOW}(\text{near32}))$$

Shifting the high 16 bits up three places moves the table index into the selector's index field. Adding 7 to the new selector value has the effect of setting the TI bit to 1 (indicating the LDT) and setting the RPL to ring three—OS/2 application privilege level. The effect of these equations are reflected graphically in Figure 3.

OS/2 2.0 currently limits all applications to the compatibility region to assure maximum compatibility between 32-bit and 16-bit code. Therefore, all applications that run under OS/2 2.0 are limited to 512 megabytes of address space, although the theoretical limit is 4 gigabytes.

Each selector is mapped to a 64K segment because the 16-bit offset registers may access only 64K of memory. The 64K boundary problem occurs when a pointer to a data structure defined by 32-bit code is passed to a 16-bit function. The 32-bit code is not limited to 64K, and the compiler is generally unaware of these 16-bit code limitations. The 32-bit compiler could possibly lay a data object across a 64K boundary, causing a portion of the structure to be inaccessible to the 16-bit code. Misplacement of such a data structure is shown in Figure 4.

To solve this problem and others related to pointers, IBM added the compiler directives, `#pragma seg16()` and `#pragma stack16()`, and a couple of new keywords, `_Far16` and `_Seg16`, to the C Set/2 compiler.

The `#pragma seg16()` directive tells the compiler to align the enclosed data object on a 64K boundary. If the object is less than 64K in size, it guarantees its placement entirely within a 64K segment. The code excerpt in Figure 5 shows how `#pragma seg16()` works. Notice that the directive works through typedefs as well as directly.

In this example, the keyword, `_Far16`, is used in the function prototypes to tell the compiler that calls to these functions should be thunked to 16-bit calls.

```
#pragma stack16(8192)           //8192 stack for far16 calls

char buffer'512 ;
#pragma seg16(buffer)           //buffer is 64K aligned

typedef struct myStruct {
    char * _Seg16 name;
    short flags;
    short time;
} MYSTRUCT;
#pragma seg16(MYSTRUCT)         // all variables of type
                                // MYSTRUCT are 64K aligned
MYSTRUCT ms;                    // ms is 64K aligned
MYSTRUCT * _Seg16 pms = ms;     // pms stored in far16 format

void _Far16 _Pascal FunctionA(MYSTRUCT *);
void _Far16 _Pascal FunctionB(MYSTRUCT **);

void main() {
    FunctionA(&ms);
    FunctionB(&pms);
}
```

Figure 5. Example of how `#pragma seg16()` works.

The function of `_Seg16` is to force storage of a pointer in far16 format. This is necessary because the compiler can not be relied upon to know that this field of the structure will be accessed by the 16-bit function when it receives a pointer to an instance of this structure as a parameter.

For instance, `FunctionA()` in Figure 5 will automatically convert the pointer to `MYSTRUCT` to a far16 pointer by virtue of the use of `_Far16` in the prototype. However, it has no way of knowing that the character pointer stored within the structure will be accessed by `FunctionA()`. It should be stored in a manner compatible with 16-bit code, using the `_Seg16` keyword.

Pointers to pointers should also be considered carefully. The compiler will generate the conversion for the pointer passed, but its value is also a pointer and should be stored in far16 format for the receiving code's benefit. In the example of Figure 5, this concept is depicted in the parameters of the call to `FunctionB()`.

Finally, since a different stack is used in 16-bit function calls, the directive `#pragma stack16()` allows the application designer to specify the size of the 16-bit stack segment. (See the sidebar "Using Borland Tools" for an explanation of the thunk support differences between Borland tools and IBM tools.)

The Difference in Call Models. The third issue that must be resolved by a thunk is that of call model differences. The 32-bit code is making near32 calls, while the 16-bit code is expecting to re-

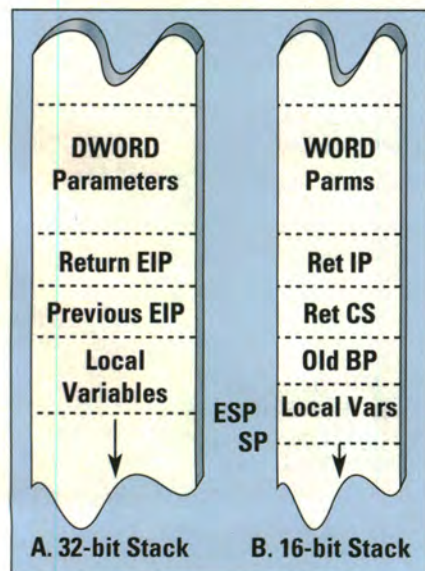


Figure 6. Near32 call and far16 call.

```
extern void foo(void);
void main() {
    foo();
}

-----

.386
.model    FLAT
assume    cs: FLAT, ds: FLAT, ss: FLAT, es: FLAT

_TEXT     segment dword public use32 'CODE'
_main     proc near
    ENTER    0,0          ; set up stack frame for main
    CALL     near ptr _foo ; call foo (underscore added)
    POP      EBP          ; restore stack frame
    RET      4            ; return from main
_main     endp
_TEXT     ends

extrn     _foo:near       ; declare foo as extern
public    _main           ; declare main as public

end
```

Figure 7. Code before minor changes.

```
extern void _Far16_Pascal foo(void);
void main() {
    foo();
}

-----

.386
.model    FLAT
assume    cs: FLAT, ds: FLAT, ss: FLAT, es: FLAT

_TEXT     segment dword public use32 'CODE' ; NOTE: 32-bit seg
_main     proc near
    ENTER    0,0          ; setup stack frame for main
    CALL     near ptr foo$32 ; call thunk layer for foo
    POP      EBP          ; restore stack frame
    RET      4            ; return from main
_main     endp

foo$32     proc far        ; thunk layer for foo
    ENTER    4,0          ; stack frame, 4 bytes local
    PUSH     EBX           ; OS/2 1.0 does not preserve
    PUSH     EDI           ; these three registers
    PUSH     ESI           ; in its 16-bit functions
    PUSH     ES
    MOV      dword ptr [EBP-4],ESP
    MOV      AX,SS         ; Setup 16-bit stack
    CMP      AX,seg FLAT:_DATA
    JE       short NO_BOUNDARY
    MOV      EAX,ESP
    CMP      AX,4096
    JAE      short NO_BOUNDARY
    XOR      AX,AX
    XCHG     ESP,EAX
    RET
```

Figure 8. Code after minor changes (continued on page 54).

turn from far16 calls. After a 32-bit near call, the stack is described by the diagram of Figure 6a. After a 16-bit far call the stack would look like the diagram of Figure 6b.

To solve this problem, the thunk is comprised of two separate segments, a 32-bit segment and a 16-bit segment. The 32-bit portion performs a far jump to the 16-bit segment, which calls the 16-bit function. When the 16-bit function returns, the 16-bit portion of the thunk performs a far jump back to the 32-bit segment, which then returns to the caller.

A NETWARE EXAMPLE

I first came upon the need to deal with mixed model code about 18 months ago while beta testing Borland C++ for OS/2. Since I work for Novell, the obvious test for the compiler would be to try to create some NetWare-aware applications using the NetWare C Interface for OS/2. Unfortunately, the NetWare C libraries had not yet caught up to the 32-bit version of IBM's PC operating system. From Novell's and its customers' standpoints, it did not make much difference because the NetWare requester for OS/2 2.0 was still entirely 16-bit code. All of the earlier 16-bit utilities would still work under OS/2 2.0. Finally, most of the OS/2 engineering team at Novell still used Microsoft C 6.0 for OS/2 1.0 to develop Novell's OS/2 NetWare utilities.

Since the NetWare requester, libraries, and DLLs were all 16-bit code, a conversion layer needed to exist at some point along the chain of communication between the 32-bit application I was attempting to build and the 16-bit requester.

Novell did a splendid job of abstracting all of the NetWare C Interface data types, defining most of those types in a single header file. To create a model-independent library, all memory locations are passed as 32-bit far (far16) pointers and all function calls are made as far calls.

This model-independent library concept turned out to be an excellent advantage to my conversion work because most of the work had been done for me. I simply redefined the abstract keyword, `NWFAR`, from `far` to `_Far16`. Then I changed all references to type `int` in the header files to `short` since, as discussed earlier, the 16-bit code expects

16-bit integer quantities passed on the stack for 'int' parameters.

After three or four changes to a single header file, I was making 16-bit library calls from a 32-bit application. The use of a few well-placed `#ifdef` directives made the library compiler and OS version independent.

Prototypes for functions in a 16-bit OS/2 1.x library typically look something like this:

```
int far FunctionA(char *a, int b);
void far FunctionB(void);
char far * far FunctionC(char *a);
```

The same prototypes adapted for a 32-bit OS/2 2.0 compiler look like this:

```
int _Far16 _Pascal FunctionA(char *a,
short b);
void _Far16 _Pascal FunctionB(void);
char * _Far16 _Pascal FunctionC(char
*a);
```

Note the use of the `_Pascal` keyword. OS/2 1.0 makes use of Pascal calling convention by default, whereas 2.0 defaults to C calling convention.

The next example demonstrates what happens when the compiler finds

```
NO_BOUNDARY:
    PUSH    EBP
    MOV     EBX,ESP
    PUSH    SS                                ; save 32-bit stack
    PUSH    EBX
    LEA     ESI,dword ptr [EBP+8]
    XOR     ECX,ECX
    SUB     ESP,ECX
    MOV     EDI,ESP
    REP MOVSB                                ; move parameters
    MOV     EAX,ESP                            ; perform address thunk
    ROR     EAX,16                            ; put seg in AX
    SHL     AX,3;                             ; move descr index
    OR      AL,7                               ; set TI and RPL bits
    ROL     EAX,16                            ; restore seg to E(AX)
    PUSH    EAX                                ; push Seg and Offset
    LSS     SP,dword ptr [ESP]                ; switch to 16-bit stack
    JMP     far ptr foo$16                    ; JUMP to 16-bit thunk

RET_F00:                                     ; return point
    SHL     EAX,16                            ; restore 32-bit stack
    SHRD    EAX,EDX,16
    MOVZX   ESP,SP
    LSS     ESP,fword ptr [ESP]
    POP     EBP
    MOV     ESP,dword ptr [EBP-4]
    POP     ES
    POP     ESI
    POP     EDI
    POP     EBX
    LEAVE                                       ; clean up locals
    RET                                        ; return to caller

foo$32
_TEXT
ends

extrn     F00:near
public   _main

assume    cs:_TEXT16

_TEXT16 segment word public use16 'CODE' ; NOTE: 16-bit seg
foo$16 proc far                            ; 16-bit thunk layer
    CALL    far ptr F00                    ; call 16-bit function
    JMP     far ptr RET_F00                ; return to 32-bit thunk
foo$16 endp
_TEXT16 ends
end
```

Figure 8. Code after minor changes (continued from page 53).

an instance of the `_Far16` keyword in a function prototype. The code fragment shown at the top of Figure 7 generates the assembly output below this fragment in the same listing. This code is straightforward.

Now look at the same fragment with one minor change at the top of Figure 8 and then the generated assembly output below it.

With this information, it should be a fairly simple process to convert most 16-bit OS/2 libraries or DLLs to a form usable by a 32-bit OS/2 compiler by tweaking the header files for the library a bit. After all, what good is a 32-bit application without access to the outside world through existing libraries? One would think this would not be much of a problem really, but I was amazed at the stumbling blocks I came across when trying to use libraries that were common to the OS/2 1.0 programmer. The information in this article should make the learning curve a little easier to handle for you than it was for me.

John Calcote is currently a course developer and instructor for advanced computer science topics at Novell's Services Division. He has been a senior consultant and programmer for Cincinnati Bell Information

Systems and an advanced support technician for Novell. He has a bachelor's degree in computer science from Brigham Young University. He can be reached via internet at jcalcote@novell.com.



REFERENCES

OS/2 2.0 Application Design Guide, IBM Inc. as part of the OS/2 Technical Library, published by QUE books.

OS/2 2.0 Programmer's Reference, IBM Inc. as part of the OS/2 Technical Library, published by QUE books.

Borland C++ for OS/2 Programmer's Reference Manual, Borland International, Inc., published by Borland International.

Borland C++ for DOS Programmer's Reference Manual, Borland International, Inc., published by Borland International.

Borland C++ for OS/2 On-Line documentation.

IBM C Set/2 On-Line documentation.

Intel486 Microprocessor Family Programmer's Reference Manual, Intel Corp., 1992.

The Award Winning, Multi-Function Enhancements for OS/2[®]

DeskMan/2[™]

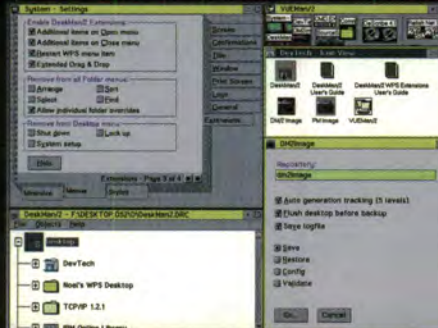
NEW VERSION 1.51
WITH WARP SUPPORT!

DeskMan/2[™] Workplace Shell[™] Manager
Efficiently Manage Your Workplace Shell!

- Easy to use drag & drop interface
- Complete command line interface + REXX & CID support
- Manage standardized or personalized desktops, customizing access to features
- Selectively save and recreate objects; even transport them between desktops, computers & versions of OS/2
- Recover from or prevent accidental changes to your desktop
- Much more - total management of the Workplace Shell

VUEMan/2[™] Virtual Desktop Manager
Increase Your Productivity!

- Makes multitasking easy; use and organize many concurrent activities with an easy to use drag & drop, point & click user interface
- Switch effortlessly between as many as 81 "virtual desktops" - like having 81 monitors that fit right on your desk!
- Drag and drop objects from WPS to virtual desktops, and between virtual desktops
- "Layouts" record window position and size for automatic repositioning and re-sizing at any future time
- Password protection for windows



All this functionality & much more
Four Great Solutions - One Low Price!

"DeskMan/2 dramatically improves the ability of corporations and users to get the most out of OS/2. DeskMan/2 helps administrators restrict and control what users are doing and provides robust backup/restore features and seamlessly enhances the features of the WPS."

Dr. Mike Kogan:

Lead Architect of OS/2 2.0
Author: The Design of OS/2

DM2Image[™] Configuration Snapshot Facility
Saves Your Working Environment!

- Save or restore an unlimited number of complete system configurations (including active or inactive WPS desktops)
- Terrific for disaster recovery; training & demo machines; other uses
- Extensible rules let you customize configuration definitions
- Configuration snapshots are compressed for maximum efficiency
- Protects WPS desktop, config.sys, startup.cmd, .INI files, autoexec.bat, WIN-OS/2 desktop & system config. data

DeskMan/2 Extensions Workplace Shell Extensions
Configure the Workplace Shell to Work Your Way!

- Modify and enhance the runtime behavior of the Workplace Shell
- Enjoy drag & drop without using special keys to shadow or copy
- Control, selectively or globally, all WPS defined object styles, menu items and capabilities - the ability to Arrange, Sort, Delete, Rename, Settings, Lockup, Shutdown, etc. (even object visibility!) is completely definable
- Corporate Edition available with password protection for objects, audit trailing for WPS activities, and other custom enhancements

Copyright 1994 Development Technologies, Inc. All rights reserved. DeskMan, DeskMan/2, VUEMan, VUEMan/2 and DM2Image are trademarks of Development Technologies, Inc. CompuServe is a registered trademark of CompuServe, Inc. Workplace Shell is a trademark of International Business Machines Corporation. IBM, OS/2 and TalkLink are registered trademarks of International Business Machines Corporation.



DevTech

Development Technologies, Inc.

Software Development & Technology Transfer
308 Springwood Road, Forest Acres, SC 29206-2113
Voice: (803) 790-9230 Fax: (803) 738-0218

Circle Reader Service Number 24

MAY / JUNE 1995

Alternative Programming Languages CD-ROM



This CD-ROM is your definitive resource for the latest in cutting-edge programming environments. Gain access to the most innovative, solution-providing languages and tools available today!



Examine the next generation of languages

You need this tool to get up to speed on distributed computing, embeddable languages, increased productivity, and languages of the future.

You get all these languages, compilers, interpreters - Perl, Glish, Smalltalk, Sather, Tcl, M0, Modula-3, ReXX, Lout, Ghostscript, Dylan, Duel, Bob, Neudl, Python, Oberon, Parasol, S-Lang, Quincy, uSystem, Distributed C, the GNU compiler suite—and more!

Put this CD-ROM to work immediately on your development projects:

- **Object-Oriented Development.** Tired of C++? Look no further than Sather, Dylan or Bob and get re-energized.
- **Distributed Computing.** When building distributed systems, languages like Parasol and M0 make your life easier.
- **Multiple Platforms.** Check out Perl and Python, plus a wide range of compilers, including the GNU compiler suite.
- **Text Processing and Document Formatting.** Lout and Ghostscript let you edit and produce complex documents with ease.
- **Prototyping.** When prototyping connected distributed systems, Glish makes it all happen for you.
- **Neural Networks.** Neudl takes the pain out of building complex neural networks.
- **And much more!**

FREE CALL!
800-431-8567
24 HOURS A DAY
7 DAYS A WEEK

All these languages are robust, imaginatively designed, break new technical ground, and come with source code for compilers or interpreters. And you also get the full descriptions and specifications.

SPECIAL BONUS!

With your purchase of this CD-ROM, you'll also receive a copy of the *Dr. Dobb's Sourcebook, Alternative Programming Languages*
A \$4.95 value — ABSOLUTELY FREE!

YES! Send me the Dr. Dobb's Alternative Programming Languages CD-ROM immediately. I will also receive the Dr. Dobb's Sourcebook Alternative Programming Languages as a bonus.

My price is \$49.95 (plus shipping: \$2.00 U.S./Canada; all other countries \$12.50). California residents, please add \$4.25 sales tax.
Clip out or photocopy this form.

Name

Address

City/State/Zip

Country

Here's how I'm paying (circle one): VISA - MC - AMEX - Check enclosed

Credit card number (include all digits)

Expiration Date

Signature required

OS/21

For more information, send E-Mail to:

Alternative Programming Languages CD
info_altcd@mfi.com

FREE CALL: 800-727-3662 24 HOURS A DAY, 7 DAYS A WEEK **FAX ORDERS: 510-372-8582**
E-MAIL ORDERS: sbarnes@mfi.com **MAIL ORDERS:** ALT CD, P.O. Box 1525, Martinez, CA 94553
INTERNATIONAL: USE MAIL, FAX 510-372-8582, E-MAIL sbarnes@mfi.com, OR CALL 415-655-4190



This article offers some translation principles to consider when designing, coding, and testing a product for international use. By BENETTA PERRY

Translated Products: Designing and Testing

Many people and activities are involved in getting a software product translated and tested. Translation entails much more than just finding someone to convert the English text into another language. Translation for Program Integrated Information (PII)¹ and publications costs in IBM can be several cents per word or more, depending on what must be translated, schedules, and the availability of translators. Plus, additional costs are often incurred because many developers do not design and code their original application for translation, that is, for future international use.

The testing of a translated product is also dependent on its design. If the product is properly designed, the testing effort is simplified, thus lowering the cost of getting the translated product to market.

In this article, we'll offer some translation principles to consider when designing, coding, and testing a product for international use.

DESIGNING A PRODUCT FOR INTERNATIONAL USE

Internationalization is the design and coding of a software product for international use. First, you should separate all code from any text that is displayed to the user. None of this text should be hard-coded or embedded within a piece of code. Translating text should not "break" the product. Avoid the use of restricted field lengths. When translated from English to other languages, text often expands in length and requires more on-screen space.

If these and other rules are not followed in the design stage, there will be changes to

the product during the testing phase. The costs of application changes during the testing phase are nearly double the cost of implementing the same changes earlier. "Translation Procedures," an IBM Information Development Guideline, advises authors how to write for efficient translation. It discusses terminology, writing style, physical factors (page layout, tags, white space, graphics), cultural factors, special concerns with online information, and other items of concern.

Developers can design a product for international use by applying the following quality translation principles during design and coding:

- Make sure artwork (faces, animals, phones, and so on) is culturally neutral.
- Avoid examples that are specific to the U.S. culture or way of doing business (banking, payroll, sports).
- Avoid illustrating idiomatic expressions specific to the U.S. English language in the text or artwork.
- Do not use humor.
- Use irony with caution.
- Avoid slang, jargon, and colloquialisms.
- Do not use personifications.
- Do not express dates in all-number formats.
- Use consistent wording; using "also called" and "synonymous with" is inconsistent. Ask coworkers to review the text looking for inconsistent terminology.
- Use clear and precise wording; avoid ambiguous words.
- Avoid grammatical errors. Use a spell checker or proofing tool to eliminate all spacing errors, misspelled words, and double words.

- Use complete sentences.
- Reduce unnecessary words; combine similar help panels where possible and appropriate.
- Avoid editing the English after translation has begun.
- Use correct references; cross-check your work. If you refer the user to a section, make sure the section title and section reference are identical. Minimize the use of hard-coding references; use tags if possible and tools if available.

If the previous guidelines are not followed, your application product may:

- Cause translators to initially translate the PII improperly, requiring them to retranslate later, doubling the cost
- Cause schedule delays because translators become confused and ask many questions, decreasing productivity
- Cause translators to translate unnecessary words, thus raising the cost of translation

- Cause these problems to be discovered late in the product cycle, forcing translators to make corrections, possibly introducing still other problems into the product.
- Cause automated translation tools to be ineffective.

Figure 1 shows a general checklist to use when designing a product for international use.

TRANSLATION PRINCIPLE CASE STUDY

Does applying these translation principles affect translation costs? In one study in which a product was translated into 15 languages, just reducing the number of words saved money. In eight help panels, unnecessary words were eliminated, cutting the word count from 476 to 288. Assuming translation costs of 50¢ per word, the cost savings from cutting unnecessary words totaled \$1,400. Imagine possible cost reductions if the other quality translation principles had been applied throughout the coding of the product.

Our analysis shows that translation costs increase when quality translation principles are applied during testing. Translation costs are likely to be lower when developers apply the translation principles as they code the product.

TESTING TRANSLATED PRODUCTS

Testing Europe/Middle East/Africa (EMEA) Translated Products on OS/2. Depending on how the application was designed, the developer of an OS/2 application could do some functional testing using the English language version of the application.

First, reconfigure OS/2 for the specific country desired. This is accomplished by simply changing the COUNTRY=, DEVINFO=KBD, and CODEPAGE= statements in the config.sys as shown in Figure 2. The parameters for these statements are listed in Table 1.

Second, reboot your system. Load the application. Any culturally sensitive functions should be displayed in the proper country format.

If an application uses Prf* APIs for OS/2 Presentation Manager, the oper-

1.	YES NO N/A	All PII and presentation control information is isolated from executable code at the source and at the load module level.
2.	YES NO N/A	Provision is made for displaying longer messages due to expansion during translation.
3.	YES NO N/A	Functions dependent on location of panel elements are not inhibited by display position changes caused by PII expansion.
4.	YES NO N/A	Development provides translators with a method for tracking messages and panels.
5.	YES NO N/A	Variables used in PII are permitted to assume any location and order within a display field.
6.	YES NO N/A	Icons and clip art are treated as PII.
7.	YES NO N/A	Inputs such as commands, keywords, and responses are treated as PII.
8.	YES NO N/A	Calendar formats are selectable.
9.	YES NO N/A	Date formats are selectable.
10.	YES NO N/A	Formats for time of day are selectable.
11.	YES NO N/A	Cardinal number shapes (Hindi, Arabic, Thai, and so on) are selectable.
12.	YES NO N/A	Mathematical/Numeric formats are selectable.
13.	YES NO N/A	Format of monetary values is selectable.
14.	YES NO N/A	System for measurement of physical quantities is selectable.
15.	YES NO N/A	Sentence spacing and punctuation is selectable.
16.	YES NO N/A	Design of such functions as sorting, searching, merging, comparing, and rounding allows for unique country and language requirements to be satisfied.
17.	YES NO N/A	Each keyboard layer is capable of having a caps lock function and a shift lock function that are independently operable.
18.	YES NO N/A	Positioning of graphics on the tops and front faces of keys must not be restricted by the manufacturing process.

Figure 1. Checklist for designing a translatable product.

```
country=001,C:\OS2\SYSTEM\COUNTRY.SYS
codepage=850,437
devinfo=KBD,US,C:\OS2\KEYBOARD.DCP
```

Figure 2. Statements in config.sys.

ating system can be configured for cultural settings in another method other than changing the parameters in the `config.sys` file.

This is done from the OS/2 Presentation Manager by changing the parameters of the Country Settings panel behind the System Setup panel. Such modification changes the corresponding entry in the `os2.ini` file.

The changes take effect when the OS/2 Presentation Manager GUI application is run.

The keyboard layout can be changed from the OS/2 command line by using the `KEYB` command followed by the keyboard ID (for example, `KEYB FR` for France).

The current code page can be changed from the OS/2 command line using the `CHCP` command (`CHCP 850` changes the current code page to 850, but `CHCP` used alone simply displays the current code page).

Testing Asia/Pacific (AP) Translated Products on OS/2. For Asia/Pacific countries, their respective operating systems, as listed in Table 2, have to be installed and then the application loaded to test for culturally sensitive functions. (IBM developers: contact your NLS Site Coordinator for procedures on obtaining these operating systems. Non-IBM developers need to go through normal IBM marketing channels to purchase these operating system versions.)

What To Look For In The Translated Product. The developer should execute all of the products' functions that he or

she has designed and coded. However, the designer does not need to configure the system for all languages. The testing department is responsible for thoroughly covering all languages and all functions. The following are some specifics to look for:

- The time format is correct for the specific country. Examples are shown in Table 3.
- The date format is correct for the specific country. Examples are shown in Table 4.
- The numerical format is correct for the specific country.
- The sorting is correct for the specific country.
- The appearance of the translation is appealing to the eye.
- The major functions of the product still work.

The formats presented in Table 3 and Table 4 are not specific to only OS/2 but are the national language industry standards. OS/2 may or may not have adopted each given format.

The initial design and coding of an application for future internationalization greatly simplifies the language translation and testing activity later on. Errors or design flaws in the original English version can greatly impact translation and testing costs as each new country version is modified and tested. National language translation is a big help in making applications ready for internationalization. For companies to remain competitive and profitable in foreign markets, however,

it is important that developers follow these design and coding guidelines.

Benetta N. Perry is a senior associate programmer at IBM Austin, Tex. In her 11 years with IBM, she has specialized in software quality assurance, testing products for functional accuracy, performance, memory, and disk usage. She has worked on OS/2, OS/2 Communications Manager, OS/2 Database Manager, OS/2 Query Manager, and the Distributed Computing Environment for OS/2. Currently she is a member of the National Language Translation & Support team working to translate and test LAN products. Benetta has a B.S. degree in computer science from Grambling State University.

OS/2 J ... Japan
OS/2 H ... Korea
OS/2 T ... Taiwan
OS/2 P ... People's Rep. China

Table 2. AP operating systems.

COUNTRY	TIME FORMAT
Sweden	h:mm:ss
Germany	h:mm:ss
Italy	h:mm:ss
Denmark	h:mm:ss
France	h:mm:ss
Norway	h:mm:ss
Spain	h:mm:ss
Japan	h:mm:ss
Korea	h:mm:ss
People's Rep. China	h:mm:ss
Taiwan	h:mm:ss
Thailand	h:mm:ss

Table 3. National time format.

COUNTRY	DATE FORMAT
France	dd/mm/yyyy
Germany	dd.mm.yyyy
Italy	dd/mm/yy
Spain	dd/mm/yy
Norway	dd.mm.yy
Japan	yyyy-mm-dd
Sweden	yyyy-mm-dd
Denmark	dd-mm-yy
Korea	yyyy.mm.dd
People's Rep. China	yyyy.mm.dd
Taiwan(Big 5)	yy/mm/dd

Table 4. National date formats.

COUNTRY	OS/2 COUNTRY CODE	OS/2 KEYBOARD ID	OS/2 PRIMARY CP	OS/2 SECONDARY CP	G kbd LAYOUT
Canada(French)	002	CF	863	850	058
Denmark	045	DK	850	—	159
France	033	FR120	850	437	120
Germany	049	GR	850	437	129
Italy	039	IT142	850	437	142
Japan	081	JP	942	437	103
Korea	082	KR	949(KS)	850,437	103
Netherlands	031	NL	850	437	143
Norway	047	NO	850	—	155
People's Rep. China	086	PR	1381	437,850	41AB
Spain	034	SP	850	437	172
Sweden	046	SV	850	437	153
U.S.	001	US	437	850	101G
Rep. of China	088	TW	950(Big 5)	437,850	103

Table 1. `config.sys` parameters.



REFERENCES/ENDNOTES

Manuals/Books

National Language Design Guide: Designing Enabled Products Vol. 1, IBM Publication Number SE09-8001-01

National Language Support: Reference Manual Vol. 2, IBM Publication Number SE09-8002-03

Information Development Guideline: Translation Procedures, IBM Publication Number ZZ27-1972-04

Articles

"IBM's Innovative Translation Process" by Lynn Denton

"An Introduction to Universal Language Support" by Lisa Abbott

Acknowledgement

I'd like to thank Veronica Bonebrake for her contribution via the IBM internal article "Country and Language Support on OS/2 and AIX."

1. Program integrated information (PII) is the text that is presented to the user online: pop-ups, pull-downs, screen prompts, panels, helps, and messages.



Invest a stamp Save a bundle

For the price of a stamp, you can get the latest edition of the federal government's free **Consumer Information Catalog** listing more than 200 free or low-cost government publications on topics such as federal benefits, jobs, health, housing, education, cars, and much more. Our booklets will help you save money, make money, and spend it a little more wisely.

So stamp out ignorance, and write today for the latest free **Catalog**. Send your name and address to:

**Consumer Information Center
Department SB
Pueblo, Colorado 81009
U.S. General Services Administration**

A public service of this publication and the Consumer Information Center of the U.S. General Services Administration.



A PROGRAMMING TOOL THAT WON'T SCREW YOUR HEAD UP

Splat! There goes another masterpiece. It was going to be a work of art, a monument to elegance and flexibility. It was going to be your finest creation. What happened? It got beaten into unrecognisable garbage by some clumsy GUI builder. As for the controls, they ended up as subtle as a flying sledgehammer thanks to a resource editor with an attitude.

This kind of mental torment could turn a decent living programmer into a serial killer. Somebody out there had better come up with a solution.

We just did. It's called Prominare, a professional's programming tool for GUI creation that has all the answers.

Take a look at the code it generates and you'll find it looks very familiar. It's exactly the way you would have written it yourself. If that sounds a bit scary, don't worry, it's real friendly. Define your naming convention and Prominare will stick to it to the letter. Prominare generates code exactly the way you want it. Your way.

If your looking to create custom controls. Prominare is a designers dream. Use it as the resource editor and you'll find that you have enough options to give you total freedom. It will also handle all major



object libraries and happily deal with PEN and multimedia. It handles all versions of OS/2 and there's no problem with understanding your old resources or Windows resources either.

Apart from being friendly, Prominare is also intelligent. When you make modifications to an application you won't be hindered by unnecessary generation phases, because it only regenerates the parts that have been modified. If you wish to make a manual modification Prominare will honour and obey it.

Now here's the best bit. We've produced a shareware version called Prominare Lite that's freely available on the Internet. Just help yourself to it and see how it beats the stink out of anything else. Then get your head around this question. If the shareware version is this good, what will the full version do?



Prominare

Tel: +1 (416) 363 2292 Fax +1 (416) 363 6157
Tel +44 (117) 972 8500 Fax +44 (117) 972 8600
e-mail: designer@interlog.com
anonymous ftp: gold.interlog.com/pub/prominare

AVAILABLE THROUGH:
INDELIBLE BLUE TEL: 1-800-776-8284 (OUTSIDE USA 1-919-878-9700)
OS/2 EXPRESS TEL: 1-800-672-5945 (OUTSIDE USA 1-612-823-6255)
EGGHEAD SOFTWARE TEL: 1-800-344-1123 (OUTSIDE USA 1-509-922-7031)
IMAGESOFT INC. TEL: 1-800-245-8840 (OUTSIDE USA 1-516-767-2233)
PROGRAMMERS PARADISE TEL: 1-800-441-1511 (OUTSIDE USA 1-908-389-9228)



Send text messages to pagers directly from OS/2!



- Easy-to-use workplace shell application
- Includes a command line interface
- *You can 'page-enable' your applications!*
- 32-bit, SOM-based code - Only \$79

ChipChat® Wireless Communicator

ChipChat-Cawthon Software
Dearborn, Michigan USA & Fukuoka Japan
Phone 313-565-4000 Fax 313-565-4001

Circle Reader Service Number 26

TCP/2™

TCP/IP for OS/2®

Our customers say our tcp/ip
is the best because
it's faster, more complete,
and reliable.
Find out for yourself!

Supported: *all* released versions of OS/2
including 32bit API for 2.x and above;
SLIP and/or PPP on COM1 and/or COM2;
Ethernet; Token Ring; ARCNet; NDIS and ODI

(800) OS2-TCP2

Essex Systems, Inc.

One Central Street, Middleton, MA 01949

A Dan Lanciani Product

TCP/2 is a Trademark of DLD consulting. All other products and trademarks are the property of their owners. TCP/2 is Based in part on work done by the University of California Berkeley.

Circle Reader Service Number 29

JOB SCHEDULING SERVER

OS/2 PM 32-bit advanced job scheduler for NOVELL and IBM LANs. Schedules multiple OS/2, DOS, and Windows jobs for concurrent execution on specified or auto selected LAN nodes. Conditional job scheduling (depends on files and other jobs return codes), security, sophisticated calendar, job logging, priority, user APIs, job recovery, periodic scheduling, and much more.

Microwork, Inc.

Phone: (708) 940-8979

Fax: (708) 940-8979

Opt-Tech Sort/Merge

High Performance Sort/Merge/Select Utility

Use as a stand-alone utility or call as a Subroutine. Many features including unlimited file size, multiple keys, record selection, duplicate elimination, summing and more!

Call for free brochure

Opt-TechData Processing
P.O. Box 678

Available for:

OS/2 or Unix \$249

DOS or Windows \$149

Zephyr Cove, NV 89448

Phone (702) 588-3737

Fax (702) 588-7576

Circle Reader Service Number 28

OS/2 Essentials \$39.95

Screen Saver

File Manager

Directory Space Grapher

Task Bar

A must have for any serious developer!

Quantity Discounts

Site Licensing Available

Stardock Systems Inc.

Phone: (313) 782-2248 Fax: (313) 782-9868

Circle Reader Service Number 30

The easy to use 32 bit OS/2 PM IPF Language Editor...

IPF Editor

- Add help to applications in minutes
- Designed for easy use by programmers and non-programmers
- Generate C and Resource source files to add help to applications automatically
- Import wordprocessing files, including WordPerfect, quickly and accurately
- Preview IPF output without compiling
- Create all hypertext links automatically
- Optional Voice Support Module
- Includes complete on-line help and documentation

IPF Editor: \$150
Voice Support Module: \$50

Orders:

1-800-IPF-7622

Information: (360) 428-5025
on compuserve at
70410,2416

Perez Computing Services
4725 Monte Vista Place
Mt Vernon, WA 98273

Circle Reader Service Number 31

Products & Services Guide ads
help you reach 36,000+
professional OS/2 Developers
cost-effectively...
1/8 page ads are
as little as \$364/insertion

➡ Call today for details: ⬅
Kristin Morgan / 212-626-2498

ORBitize™

Build and Browse IDL with ORBitize™

ORBitize™ eases the development of distributed objects using an object request broker (ORB). It provides an intuitive graphical user interface for creating and browsing OMG IDL definitions rapidly. ORBitize™ browsing capabilities let you easily view and understand IDL definition hierarchies and quickly navigate between definitions. ORBitize™ eliminates the need to learn IDL syntax, allowing new developers to quickly produce error-free IDL.

ORBs: SOM, ObjectBroker, Orbix, TME, and others
Platforms: OS/2, Windows, NT, UNIX

74 Northeastern Blvd., Ste. 8B,

Nashua, NH 03062

v: 603-594-0525 f: 603-594-0527

email: info@netlinks.com



Circle Reader Service Number 32

Data Acquisition for OS/2

Presenting IOPRO/VX. This toolkit allows you to unleash the 32 bit multitasking power of OS/2 and the incredibly easy to use development environment of VX-REXX against any and all of your data acquisition or process monitoring and control applications! Plug in your favorite I/O adapter, drag and drop to build your app, and that's it. Or, create your own screen instrument panels and remotely control your instrument via RS-232, IEEE-488, or a network!

Call or write for a free brochure.

"If it's done right, chances are it's Rock Solid."

Rock Solid Software

8460 Plank Road Box 228

Montville, OH 44064

(216) 390-0590

OS/2 is a registered trademark of International Business Machines Corporation.
VX-REXX is a trademark of Watcom International Corp.

Circle Reader Service Number 35

FAX Developer Tools

From the developers of *FaxWorks for OS/2*
Client/Server API and Printer Driver Toolkits
Support for LANs, up to 96 lines per CPU,
and all popular fax hardware

Keller Group Inc.

Voice: (612) 429-7273

Fax: (800) 329-3293

Email: kgroup@ibm.net

Fax: (612) 653-1987

Faxworks is a trademark of Global Village Communication.

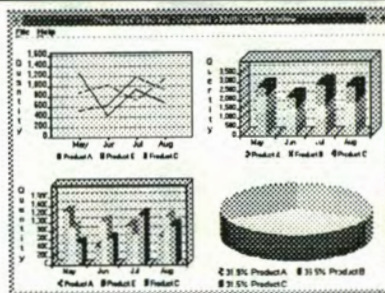
Circle Reader Service Number 33

SUBSCRIBE TODAY!

Only \$39.95 for a full year subscription to the premier source of tools and techniques for the OS/2 software developer: **OS/2 DEVELOPER**

Call today and don't miss a single issue:

1-800-WANT-OS2



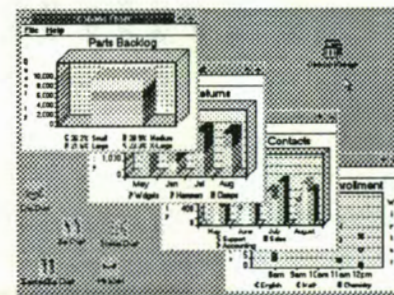
It's easy to integrate business graphics into your OS/2 PM* or Windows 3.x* application

Developer's Business Graphics Toolkit

- High-level APIs
- Tactile feedback
- Real-time charting
- 16-bit and 32-bit
- Dynamic Link Library
- Positive/negative data
- Printer/metafile support
- 25+ chart types



The Crossley Group
P.O. Box 921759
Norcross, GA 30092
USA (404) 751-3703
Int'l 44-932-844-281
(demo available)



Circle Reader Service Number 34

July/August Issue Theme:

Object-Oriented Programming

Bonus Distribution:

OS/2 World & Object World

AD CLOSE DATE / MAY 4

Invest a stamp



Save a bundle

For the price of a stamp, you can get the latest edition of the federal government's free **Consumer Information Catalog** listing more than 200 free or low-cost government publications on topics such as federal benefits, jobs, health, housing, education, cars, and much more. Our booklets will help you save money, make money, and spend it a little more wisely.

So stamp out ignorance, and write today for the latest free **Catalog**. Send your name and address to:

**Consumer Information Center
Department SB
Pueblo, Colorado 81009**

A public service of this publication and the Consumer Information
Center of the U.S. General Services Administration.

ADVERTISER INDEX

Reader Service	Company Name	Page
12	Athena Design.....	21
26	ChipChat.....	62
22	Compuware.....	47
34	Crossley Group.....	63
24	Development Technologies.....	55
29	Essex.....	62
38	Hockware.....	COV4
7	IBM.....	13
10	Indelible Blue.....	17
2	Inmark.....	1
20	JBA.....	35
3	John Wiley.....	2
33	Keller Group.....	63
17	Kovsky.....	33
36	MHR.....	67
1	Metaware.....	COV2
18	Micro Burst.....	34
6	MicroEdge.....	11
—	Microwork.....	62
8	Mitnor.....	15
32	Netlinks.....	63
5	One Up.....	7
28	Opt-Tech.....	62
31	Perez.....	62
4	Programmer's Paradise.....	4
25	Prominare.....	61
35	Rock Solid.....	63
14	Quadron.....	23
37	Rimstar.....	COV3
13	Serena.....	22
19	Softbridge.....	34
15	Stac.....	25
30	Stardock.....	62
16	Suitable Alternatives.....	29
9	Tech Bridge.....	16
21	Watcom.....	43
11	Zinc.....	19

Return these cards to
 receive **FREE** information on
 the products advertised in
 OS/2 Developer. Simply
 circle the reader service
 numbers of the items you
 are interested in, complete
 the address information,
 and mail the card.

To enter a paid subscription
 to OS/2 Developer, check
 the "YES" box, sign and
 date the card, indicate a
 method of payment, and
 drop the card in any mailbox.

Return postage is paid.

Subscribe to OS/2 Developer

☐ **Yes** I want to start my subscription to OS/2 Developer for only \$39.95!

Signature _____ Date _____

☐ Charge my credit card: ☐ Visa ☐ Mastercard ☐ American Express

Card No. _____ Exp. Date _____

☐ Bill me.

* Canada and Mexico add \$16.00; Outside North America add \$30.00 for shipping.

Name _____

Title _____

Company _____

Address _____

City _____ State _____ Zip _____

Phone Number _____ Fax Number _____

Subscribe to OS/2 Developer

☐ **Yes** I want to start my subscription to OS/2 Developer for only \$39.95!

Signature _____ Date _____

☐ Charge my credit card: ☐ Visa ☐ Mastercard ☐ American Express

Card No. _____ Exp. Date _____

☐ Bill me.

* Canada and Mexico add \$16.00; Outside North America add \$30.00 for shipping.

Name _____

Title _____

Company _____

Address _____

City _____ State _____ Zip _____

Phone Number _____ Fax Number _____

Subscribe to OS/2 Developer

☐ **Yes** I want to start my subscription to OS/2 Developer for only \$39.95!

Signature _____ Date _____

☐ Charge my credit card: ☐ Visa ☐ Mastercard ☐ American Express

Card No. _____ Exp. Date _____

☐ Bill me.

* Canada and Mexico add \$16.00; Outside North America add \$30.00 for shipping.

Name _____

Title _____

Company _____

Address _____

City _____ State _____ Zip _____

Phone Number _____ Fax Number _____

Free Information

Circle the reader service number of those items of interest and drop this card in the mail.
 Your information will be sent directly to you. See Index for Reader Service listing.

1	11	21	31	41	51	61	71	81	91	101	111	121	131	141	151	161	171	181	191
2	12	22	32	42	52	62	72	82	92	102	112	122	132	142	152	162	172	182	192
3	13	23	33	43	53	63	73	83	93	103	113	123	133	143	153	163	173	183	193
4	14	24	34	44	54	64	74	84	94	104	114	124	134	144	154	164	174	184	194
5	15	25	35	45	55	65	75	85	95	105	115	125	135	145	155	165	175	185	195
6	16	26	36	46	56	66	76	86	96	106	116	126	136	146	156	166	176	186	196
7	17	27	37	47	57	67	77	87	97	107	117	127	137	147	157	167	177	187	197
8	18	28	38	48	58	68	78	88	98	108	118	128	138	148	158	168	178	188	198
9	19	29	39	49	59	69	79	89	99	109	119	129	139	149	159	169	179	189	199
10	20	30	40	50	60	70	80	90	100	110	120	130	140	150	160	170	180	190	200

May / June 1995; Expires November 1995

RSCF95

Free Information

Circle the reader service number of those items of interest and drop this card in the mail.
 Your information will be sent directly to you. See Index for Reader Service listing.

1	11	21	31	41	51	61	71	81	91	101	111	121	131	141	151	161	171	181	191
2	12	22	32	42	52	62	72	82	92	102	112	122	132	142	152	162	172	182	192
3	13	23	33	43	53	63	73	83	93	103	113	123	133	143	153	163	173	183	193
4	14	24	34	44	54	64	74	84	94	104	114	124	134	144	154	164	174	184	194
5	15	25	35	45	55	65	75	85	95	105	115	125	135	145	155	165	175	185	195
6	16	26	36	46	56	66	76	86	96	106	116	126	136	146	156	166	176	186	196
7	17	27	37	47	57	67	77	87	97	107	117	127	137	147	157	167	177	187	197
8	18	28	38	48	58	68	78	88	98	108	118	128	138	148	158	168	178	188	198
9	19	29	39	49	59	69	79	89	99	109	119	129	139	149	159	169	179	189	199
10	20	30	40	50	60	70	80	90	100	110	120	130	140	150	160	170	180	190	200

May / June 1995; Expires November 1995

RSCF95

Free Information

Circle the reader service number of those items of interest and drop this card in the mail.
 Your information will be sent directly to you. See Index for Reader Service listing.

1	11	21	31	41	51	61	71	81	91	101	111	121	131	141	151	161	171	181	191
2	12	22	32	42	52	62	72	82	92	102	112	122	132	142	152	162	172	182	192
3	13	23	33	43	53	63	73	83	93	103	113	123	133	143	153	163	173	183	193
4	14	24	34	44	54	64	74	84	94	104	114	124	134	144	154	164	174	184	194
5	15	25	35	45	55	65	75	85	95	105	115	125	135	145	155	165	175	185	195
6	16	26	36	46	56	66	76	86	96	106	116	126	136	146	156	166	176	186	196
7	17	27	37	47	57	67	77	87	97	107	117	127	137	147	157	167	177	187	197
8	18	28	38	48	58	68	78	88	98	108	118	128	138	148	158	168	178	188	198
9	19	29	39	49	59	69	79	89	99	109	119	129	139	149	159	169	179	189	199
10	20	30	40	50	60	70	80	90	100	110	120	130	140	150	160	170	180	190	200

May / June 1995; Expires November 1995

RSCF95

Free Information



BUSINESS REPLY MAIL

FIRST CLASS MAIL

PERMIT NO. 612

PITTSFIELD, MA

POSTAGE WILL BE PAID BY ADDRESSEE

OS/2 Developer

P. O. BOX 5252

PITTSFIELD MA 01203-9317

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



Free Information



BUSINESS REPLY MAIL

FIRST CLASS MAIL

PERMIT NO. 612

PITTSFIELD, MA

POSTAGE WILL BE PAID BY ADDRESSEE

OS/2 Developer

P. O. BOX 5252

PITTSFIELD MA 01203-9317

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



Free Information



BUSINESS REPLY MAIL

FIRST CLASS MAIL

PERMIT NO. 612

PITTSFIELD, MA

POSTAGE WILL BE PAID BY ADDRESSEE

OS/2 Developer

P. O. BOX 5252

PITTSFIELD MA 01203-9317

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES





The staff of OS/2 Developer put together this special section to guide you through the various software tools specifically for GUI development. The products described in this guide are based on surveys sent to vendors; they are provided as a service to you and are free to vendors. These listings do not represent an endorsement by OS/2 Developer. Although every effort has been made to ensure accuracy, we do not assume any responsibility for error or omission in this section.

Gui Development Buyer's Guide

COMPUTER ASSOCIATES INTERNATIONAL INC.

Circle No. 101

CA-Realizer 2.0, compatible with any system running OS/2 2.1, is a visual programming environment that handles the mechanics of event-driven programming, message passing, and process sharing. It combines visual development tools, Programmable Application Tools, and a structured superset of BASIC extended to access Windows and OS/2 objects and resources. ODBC support and Xbase file access make the development of database-oriented applications and client/server solutions possible. Features include a multiplatform functionality and FormDev, which allows developers to design the visual elements of a program using standard graphical objects while automatically generating database query and entry forms directly from existing Xbase files. Multiple forms may be edited simultaneously, and application files may be maintained together as an integrated project. Price: \$247.

CA-Realia II Workbench 2.0 provides a GUI workstation and mainframe-compatible COBOL development environment on the PC to improve COBOL applications that operate under Windows and OS/2. It includes a high-speed COBOL compiler, an interactive debugger, a COBOL-Intelligent analyzer, a CICS emulator, an integrated editor, and a complete life cycle manager with mainframe connectivity. The CA-Realia II Workbench is compatible with MS-DOS 3.3 or higher, Windows 3.1, or OS/2 2.1. Price: \$2,500.

CA-SuperProject 3.0 for OS/2 is a native 32-bit application that exploits such OS/2 strengths as the Workplace shell, preemptive multitasking, and enhanced memory management. As a result of its compatibility across four platforms—OS/2, Windows, DOS, and VAX/VMS—a file saved on one version can be directly opened on another. For \$59, CA-SuperProject users can upgrade to the full ver-

sion of CA-Realizer, which provides a run time for royalty-free distribution of applications to any users. Major features include the Help Assist mode; Project Manager's Assistant; four outlines with fully integrated Gantt and histogram charts and crosstabs, flexible PERT, and WBS charts; WYSIWYG reporting with unlimited layouts, preview, user-defined fields, and formulas; and resource definition, shift and overtime management, assignment allocation, efficiency factors, materials management, flexible rate changes, overhead, cost accounts, C/SSR reporting, and cost charting. SuperProject also provides tracking and replanning, strong multi-project management, LAN support, function-level password security, import/export, macros, and multiplatform support. Price: \$649.

Computer Associates International Inc., One Computer Associates Plaza, Islandia, N.Y. 11788-7000, (800) CALL CAI or (516) 342-5224, fax (516) 342-5734.

EASEL CORP.

Circle No. 102

Object Studio is an object-oriented toolset for rapid development of large-scale production client/server applications. Object Studio includes Synchronicity 2.1 for business object modeling, auto-generation of Smalltalk code, and persistent object mapping. In addition, Object Studio contains ENFIN Smalltalk 4.1 for visual design and connectivity options and TeamBuilder 1.0 for group development. Price: \$2,995 depending on platforms and options.

Synchronicity 2.1 is a visual development toolset for maximizing the productivity gains of object technology through the design and management of business objects. Developers build high-level business models providing the foundation for applications. Smalltalk code is automatically generated through the modeling tool. The mapping tool quickly links models to relational databases. Price: \$1,995 depending on platforms and options.



Enfin Smalltalk 4.1 is a comprehensive, visual Smalltalk development environment within the Object Studio family of object-oriented visual development tools for building large-scale production client/server applications. ENFIN supports the industry's range of connectivity options and facilitates access to all major relational database management systems. Price: \$2,995 depending on platforms and options.

TeamBuilder 1.0 is a member of the Object Studio family of object-oriented visual development tools for building large-scale production client/server applications. TeamBuilder enables groups of developers to simultaneously build object-oriented applications. A developer can group objects together as a project while each object maintains a separate revision level, date-stamp, and label. Price: \$495 depending on platforms and options.

Easel Corp., 25 Coporate Dr., Burlington, Mass. 01803, (800)-OBJECTS or (617) 221-2100, fax (617) 221-3099.

FLEXUS

INTERNATIONAL CORP. Circle No. 103

COBOL spII 3.1 allows developers to implement advanced GUI screens in their COBOL programs using standard COBOL CALL statements. COBOL spII supports transparent migration of screens and COBOL source code across different GUIs, across different operating systems, and from character mode environments to graphical environments, such as OS/2 Presentation Manager. Price: \$1,000.

Flexus International Corp., P.O. Box 640, Bangor, Pa. 18013, (610) 588-9400, fax (610) 588-9475.

HOCKWARE INC.

Circle No. 104

VisPro/C 1.1 is a full-featured OS/2 2.x visual programming tool for the IBM C-Set and C-Set ++ First Step compilers. It provides drag-and-drop programming for creating multithreaded, 32-bit graphical user interface applications. VisPro/C includes Workplace Shell integration, CUA '91 objects, a graphical class browser, a visual DB2/2 database designer, and a SOM-based object builder. The new version includes code generation improvements based on recent customer feedback. Price: \$399.

VisPro/C++ 1.1 is an OS/2 visual programming tool for the IBM C-Set++ compiler and User Interface Class Library. It has all the features of VisPro/C 1.1. Price: \$399.

VisPro/REXX Bronze Edition 2.11 has an easy-to-use drag-and-drop environment that allows you to quickly develop 32-bit OS/2 2.x graphical user interface applications. It includes many of the CUA '91 objects, full Workplace Shell integration, multiple development views, and a

SOM-based object builder. The Bronze edition provides standalone, royalty-free executable. The newest version features an improved user interface, new features for manipulating window frames, a more intelligent event editor, and a new, lower price. Upgrade to the Gold edition is available. Price: \$59.

VisPro/REXX Gold Edition 2.11 is a professional drag-and-drop visual programming tool for 32-bit OS/2 2.x graphical user interface development. It includes all CUA '91 objects plus 3-D business graphics and multimedia, full Workplace Shell integration, multithread support, a visual DB2/2 database designer, multiple development views, a SOM-based object builder, and a standalone, royalty-free executable. This edition features new notebook events and APIs, improved container column and API features, and support for programming-shared memory. Price: \$299.

Hockware Inc., 315 N. Academy St. Ste. 100, Cary, N.C. 27513, (919) 380-0616, fax (919) 380-0757.

IBM

Circle No. 105

IBM VisualAge 2.0 combines object-oriented programming with the visual connection of prefabricated software components or parts. According to the company, VisualAge facilitates the creation of industrial-strength line-of-business applications for the client/server environment. VisualAge Team, for \$4,995, includes features such as COBOL language, communications/transaction, multidatabase, multimedia, and AS/400. Price for VisualAge: \$2,495.

IBM, 3039 Cornwallis Rd., Research Triangle Park, N.C. 27709, (800) 426-2279 or (919) 254-4760, fax (919) 254-4820.

INFERENCE CORP.

Circle No. 106

ART*Enterprise 1.0 for OS/2 is a client/server development tool that integrates databases, document repositories, and knowledge bases. It is fully object-oriented and includes an extensive GUI class library and painter, a rule engine, and case-based retrieval capabilities. It is suited to applications that help front office employees interface more intelligently with customers and prospects. Price: \$7,995 including one week of training.

Inference Corp., 101 Rowland Way, Novato, Calif. 94945, (800) 336-9923 or (415) 899-0100, fax (415) 899-9080.

INMARK

Circle No. 107

zApp Developer's Suite 2.2 facilitates the writing of commercial-quality applications that run on multiple platforms. It contains zApp, a C++ application framework; zApp Factory, a visual designer and code generator for the zApp environment; and the zApp Interface Pack, a set of custom controls for zApp.

Applications generated using the zApp Developer's Suite are single-source portable to 14 platforms. Price: \$1,295.

Inmark, 2065 Landings Dr., Mountain View, Calif. 94043, (800) 346-6275 or (415) 691-9000, fax (415) 691-9099.

MICRO DATA BASE SYSTEMS INC.

Circle No. 108

Object/1 3.0 is a pure object-oriented graphical development environment for OS/2 Presentation Manager 2.x or Windows. Object/1's Windows Painter, source code management, and debugging tools leverage application development productivity. Object/1 is written in Object/1, and its source code is included among more than 325 classes and 7,000 methods provided with the product. Object/1's programming tools include a forms painter, various browsers, a workspace, notifier, breakpoint, and debugger tools. The price of Object/1 includes the development system, Application Distribution System, a relational database system (TBL), and the choice of an interface to MDBC IV or DB2/2, or in Windows to SQL Server or Sybase. Price: \$4,000.

Micro Data Base Systems Inc., P.O. Box 2438, 1305 Cumberland Ave., West Lafayette, Ind. 47906-0438, (800) 445-6327 or (317) 463-7200, fax (317) 463-1234.

MICROFOCUS

Circle No. 109

Dialog System 2.5 is a complete development environment enabling COBOL programmers to rapidly build GUI client/server database applications for multiple platforms, including OS/2, Windows, and DOS. This product provides visual programming facilities that enable programmers to build, manage, and debug client/server applications using Dialog's trace and debug environment. It now includes a complete ANSI standard relational database from XDB Systems, allowing applications to be ported to any XDB system or equivalent relational database through ODBC connectivity. Price: \$1,250.

MicroFocus, 2465 East Bayshore Rd., Palo Alto, Calif. 94303, (415) 856-4161, fax (415) 856-6134.

OPEN SOFTWARE ASSOCIATES INC.

Circle No. 110

Open UI 3.1 is a Distributed User Interface Management System (D-UIMS) for enterprise-scale distributed applications. It enables the user interface of an application to be developed once and deployed on any standard GUI, including Windows, OS/2 Presentation Manager, Motif, and Macintosh, and on terminals. Open UI's message-based architecture provides many unique features to address scalability issues for enterprise-wide client/server applications. Price: \$3,500 and up.

Open Software Associates Inc., 20 Trafalgar Sq., Ste 500, Nashua, N.H. 03063, (603) 886-4330, fax (603) 598-6877.

PARCPLACE SYSTEMS INC. Circle No. 111
VisualWorks 2.0 is a client/server tool for building portable applications using object-oriented technology. A Database Application Creator is included for rapid application development. Applications developed are instantly portable across multiple platforms, are scalable across the enterprise, and can have their functionality distributed between both clients and servers. Price: \$2,995 for OS/2, Windows, Windows NT, and Macintosh; \$4,995 for UNIX.

ParcPlace Systems Inc., 999 East Arques Ave., Sunnyvale, Calif. 94086-4593, (800) 759-7272 or (408) 481-9090, fax (408) 481-9095.

SCIENTIFIC ENDEAVORS CORP. Circle No. 112

Graphic/OS2 7.0 is a library of C routines for creating publication-quality technical graphics. The interface is designed so it is possible to write a full-featured Presentation Manager (PM) application without having to make any PM APT calls. Routines are provided for getting prompted user input and displaying text in a scrolling text window. Price: \$495.

Scientific Endeavors Corp., 508 North Kentucky St., Kingston, Tenn. 37763, (800) 998-1571 or (615) 376-4146, fax (615) 376-1571.

VISIBLE SYSTEMS CORP. Circle No. 113

Visible Analyst Workbench 5.4 is a set of CASE tools designed for large, complex applications. It automates structured analysis, design, data modeling, and enterprise modeling. Version 5.4 includes an enhanced PowerBuilder Interface, new UNIFACE bidirectional interface support, and enhancements to its repository customization capabilities. Price: \$2,795.

Visible Systems Corp., 300 Bear Hill Rd., Waltham, Mass. 02154, (617) 890-2273, fax (617) 890-8909.

WATCOM INTERNATIONAL CORP. Circle No. 114

Watcom C/C++ 10.0 is a professional, multiplatform C and C++ development system for 32-bit DOS, Windows NT, Win32s, OS/2 2.x, OS/2 Warp, and Novell NLMS, as well as 16-bit DOS and Windows 3.x. Combining its optimizing compiler technology with a new integrated development environment (IDE) and a comprehensive set of tools, Watcom C/C++ includes an advanced GUI debugger, C++ class browser, and profiler. Watcom C/C++ delivers support for C++ templates, exception han-

dling, and Microsoft Foundation Class (MFC) libraries. Price: \$199.

Watcom VX-REXX 2.1 is an integrated environment for developing OS/2 Presentation Manager applications, including a project management facility, visual designer, and debugger. This enhanced version contains notebooks, containers, sliders, pop-up menus, and DDE objects. Price: \$99.

Watcom VX-REXX Client/Server Edition 2.1 is a visual development environment for OS/2. Connection, query, and chart objects allow you to access several databases, manipulate data, and chart the results quickly and easily. Features include drag-and-drop programming; bound controls; and professional multithreaded, multiwindowed, and drag-and-drop enabled application development. Price: \$299.

Watcom, 415 Phillip St., Waterloo, Ont., Canada. N2L 3X2, (800) 265-4555 or (519) 886-3700, fax (519) 747-4971.

XVT SOFTWARE INC. Circle No. 115

XVT Development Solution for C 4.0 (DSC) pairs XVT-Design, a visual GUI layout, prototyping, and code generation tool with the XVT Portability Toolkit, a thin-layered API. XVT-Design generates XVT Portability Toolkit compatible C source

code, application resources, and makefiles. Applications developed with DSC will have native look-and-feel and single-source portability to seven different GUIs and over 30 hardware platforms. Supported GUIs include: OS/2, Windows, Windows NT, Macintosh, OSF/Motif for UNIX and VMS, and Character Screens for DOS, UNIX, and VMS. Price: \$2,325.

XVT Development Solution for C++ 3.2 (DSC++), featuring XVT-Power++, is an application framework for developing object-oriented GUI applications for over 20 platforms. XVT-Power++ uses the Model-View-Controller paradigm to provide high-level functionality, such as automatic data propagation and environment inheritance, plus fully integrated Rogue Wave data structures. DSC++ also includes XVT-Architect, a visual application builder used for the entire application development life cycle, from design to layout to final application building. DSC++ allows developers to create single-source applications with native look-and-feel on OS/2, Windows, Windows NT, Macintosh, and OSF/Motif. Price: \$2,325.

XVT Software Inc., 4900 Pearl East Circle, Boulder, Colo. 80301, (800) 678-7988 or (303) 443-4223, fax (303) 443-0969.

Take control of your OS/2 scheduling needs with
Advanced Task Scheduler for OS/2

September						
			1	2	3	
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

ATS
Version 3.0

Are you looking for a way to manage your production job streams? Do you have programs that need to run after the completion of one or more other programs? Do you want to run programs on a regular basis? Do you want to process files that have just been received from another system or modified locally? Do you need to take action if a file is missing? Do you need to schedule around holidays and periods of heavy CPU load?

With ATS you can run your programs when YOU want them to run with out you having to be there.

Here are some of the features of ATS for OS/2:

- * Manage Production Job Streams
- * Programs can be scheduled to run anytime.
- * Programs can be dependent upon Files, Other Programs, Holiday Schedules, and External Signals
- * Integrated Operators Console
- * Logging - To File and Console

New Features:

- * Job Queues - for managing your resources while managing your tasks
- * Periodic Scheduling
- * COMPLETE API and Command Line Interface

Upgrades Available from Version 2 and other Task Schedulers
Call or E-Mail for Details

MHR
Software and Consulting
2227 U.S. Highway #1 * Suite 146
North Brunswick, NJ 08902
Voice: (908) 821 - 0359 * Fax: (908) 821 - 0350 * CompuServe 70312,627

Only \$349

Circle Reader Service Number 36



Spotlight: OS/2 Library

This list was provided by Gail Ostrow at IBM's Independent Vendor League (IVL). The IVL provides support for OS/2 authors and publishers. Gail can be reached via e-mail at gailo@vnet.ibm.com or by fax at (203) 368-6379.

Warped Books

NOTE: In March 1995, John Wiley & Sons Inc. purchased Van Nostrand Reinhold's (VNR) OS/2 computer book list.

OS/2 Warp Books

The following books were written or upgraded for OS/2 Warp. All categories (Novice, Intermediate, and Advanced) are shown.

The Art of OS/2 Warp Version 3 C Programming, 0-471-08633-9, Panov et al. John Wiley & Sons
Complete Idiot's Guide to OS/2 Warp, 1-56671-589-9, Alpha
Connecting with LAN Server 4.0, 1-56276-270-2, Nance, Ziff-Davis
The Design of OS/2, 2nd Edition, 0-201-52886-X, Kogan & Dietel, Addison-Wesley
Developing High-Powered OS/2 Warp Applications, 0-471-11586-X, Reich, John Wiley & Sons
Developing Multimedia Applications Under OS/2, 0-442-01929-7, Lopez, VNR
Dynamic Data Development for OS/2, 0-442-01949-1, Puchtel, VNR
A Guide to Learning the Internet with OS/2 Warp, 1-56884-465-4, IBM Press (IDG)
Inside OS/2 Warp Version 3, 1-56205-378-7, Minasi, New Riders
Internet Connections with OS/2 Warp, Ivens, Prima
Lotus Notes Version 3 in the OS/2 Environment, 0-442-01890-8, Walsh, VNR
Making OS/2 Work for You: Installing, Configuring & Using OS/2 Warp, 0-471-06083-6, Azzarito & Green, John Wiley & Sons
Mastering OS/2 Warp Version 3, 0-7821-1663-9, Dyson, Sybex
Navigating the Internet with OS/2, 0-672-30719-7, Tyson, Sams
Official Guide to Using OS/2 Warp, 1-56884-466-2, IBM Press (IDG)
OS/2 C++ Class Library: Power GUI Programming with C Set++, 0-442-01795-2, Law et al., VNR
OS/2 for Dummies, 2nd Edition (Warp Version 3), 1-56884-205-8, Rathbone, IDG Books
OS/2 Presentation Manager GPI, 2nd Edition, 0-442-01939-4, Winn, VNR
OS/2 Programmer's Desk Reference, Gopaul, McGraw-Hill
OS/2 Remote Communications: Asynchronous to Synchronous Tips and Techniques, 0-442-01814-2, Stonecipher, VNR
The OS/2 Survival Kit, 0-201-40915-1, Proffit, Addison-Wesley
OS/2 Warp Answers, 0-0788-5-0 Ivens, Osborne/McGraw-Hill
The OS/2 Warp Bible, 1-55755-268-X, Albrecht & Plura, Abacus
OS/2 Warp Control Program API, 0-471-03887-3, Stock, John Wiley & Sons
OS/2 Warp: Easy Installation Guide, Kamin, Prima
OS/2 Warp Presentation Manager API, 0-471-03873-3, Stock & Barnum, John Wiley & Sons
OS/2 Warp Presentation Manager Programming for Power Programmers, John Wiley & Sons, 0-471-05839-4, Stern & Morrow
OS/2 Warp v.3 Unleashed, Deluxe Edition, 0-672-30595-3, Moskowitz et al., Sams
The OS/2 Warp Version 3 Goldmine, Patton, VNR
OS/2 Warp Version 3 Presentation Manager Mentor, 0-442-01989-0, Drapkin, VNR
OS/2 Warp Workplace Shell API, 0-471-03872-5, Pollack & Stock, John Wiley & Sons
The Photo CD Book (w/disk with OS/2 Warp applications), 1-55775-195-2, von Bulow & Paulissen, Abacus
Programming the OS/2 Warp Version 3 GPI, 0-471-10718-2, Knight & Ryan, John Wiley & Sons



Running Windows Applications in OS/2: A Power User's Guide, 0-442-01924-6, Anise et al, VNR
Secrets of the OS/2 Masters, 0-442-01991-2, Sullivan, VNR
Stepping up to OS/2 Warp, 1-55755-269-X, Albrecht & Plura, Abacus
Teach Yourself OS/2 Warp v.3 in a Week, 0-672-30684-0, Sams
Your OS/2 Warp Consultant, 2nd Edition, 0-672-30484-8, Tyson, Sams
Using the OS/2 Warp BonusPak, 1-55755-285-1, Hoff, Abacus
Using OS/2 Warp Version 3, Special Edition, 0-7897-0088-3, Clifford et al, QUE
The Warp Book: Your Definitive Guide to Installing and Using OS/2 v3, 0-7615-0034-0, Sosinsky, Prima
Warping to the Internet, 1-55755-284-3, Salomon, Abacus

Other OS/2 Programming Books

The following OS/2 programming books were not specifically written for OS/2 Warp but are appropriate for OS/2 2.X and Warp versions.

Advanced OS/2 Presentation Manager Programming, 0-471-59198-X, Burge & Celi, John Wiley & Sons
The Art of OS/2 C Programming, 0-471-58802-4, Panov et al., Wiley-QED
C & C++ Programming in the OS/2 Environment, 0-442-01240-3, Gopaul, VNR
Client/Server Programming with OS/2 2.1, 3rd Edition, 0-442-018333-9, Orfali & Harkey, VNR
A Client-Server Survival Guide With OS/2, 0-442-01798-7, Orfali & Harkey, VNR
Cross Platform Programming in OS/2, 0-07-017862-3, Dorfman, McGraw-Hill
Designing OS/2 Applications, 0-471-58889-X, Reich, John Wiley & Sons
Effective Multithreading in OS/2, 0-070017841-0, Dorfman, McGraw-Hill/PC
Instant OS/2 - Porting C Applications to OS/2, 0-8306-4522-5, Dorfman, McGraw-Hill
GUI-OOUI Wars: Windows Vs. OS/2, 0-442-01750-2, Mandel, VNR
Object Oriented Programming Using SOM & DSOM, 0-442-01948-3, Lau, VNR
Objects for OS/2, 0-442-01738-3, Danforth et al., VNR
The OS/2 2.1 Application Programmer's Guide, 0-442-01736-7, Kelly et al., VNR
The OS/2 2.1 Corporate Programmer's Handbook, 0-442-01598-4, Scholin, et al., VNR
OS/2 2.1 Programming, 0-07-881910-5, Schildt & Goosey, Osborne McGraw-Hill
OS/2 2.1 Workplace Shell Programming, 0-679-79162-0, Random-House
OS/2 2.X Notebook (Best of OS/2 Developer), 0-442-01522-4, Conklin, VNR
OS/2 Batch Files To Go, 0-070052370-3, Richardson, McGraw-Hill/Windcrest
OS/2 Lan Server: The Do-it-Yourselfer's Guide to Perfect Networking, 0-442-01958, Scherer, VNR
OS/2 and Netware Programming: Using the Netware Client API for C, 0-442-01815-0, Adair et al., VNR
OS/2 PM Programming for COBOL Programmers, Revised Edition, 0-471-56140-1, Chapman, Wiley-QED
OS/2 Presentation Manager Programming, 1-56276-123-4, Petzold, Ziff-Davis
OS/2 Presentation Manager Programming: Hints & Tips, 0-07-70776-8, Goodyer, McGraw-Hill
OS/2 V2 C++ Class Library: Power GUI Programming with C Set++, 0-442-01795-2, Law et al., VNR
Quick Reference Library for OS/2 Functions: Vol 1 - Win Functions, 0-442-018975, Scholin, VNR
Quick Reference Library for OS/2 Functions: Vol 2 - Message Functions, 0-442-018983, Scholin, VNR
Real World Programming for OS/2 2.1, 0-672-30300-0, Blain et al., Sams
Real World Programming for OS/2 2.11, 0-672-30563-1, Blain et al., Sams
The Ultimate OS/2 Programmer's Manual, 0-07-043972-9, Mueller, McGraw-Hill
Writing OS/2 Device Drivers in C, 2nd edition, 0-442-01729-4, Mastrianni, VNR

REXX Programming Books

Application Development Using OS/2 REXX, 0-471-60691-X, Rudd, Wiley-QED
Mastering OS/2 REXX, 0-471-51901-4, Gargiulo, Wiley-QED
OS/2 2.1 REXX Handbook, 0-442-01734-0, German, VNR
REXX: Advanced Techniques for Programmers, 0-07-034600-3, Kiesel, McGraw-Hill
The REXX Cookbook: A Tutorial Guide to REXX in OS/2 for the IBM PC, 0-9632773-4-0, Callaway, Sams
REXX Reference Summary Handbook, 2nd Edition, 0-9639854-2-6, Goran, CFS Nevada
REXX Tools and TEchniques, Nimal, John Wiley & Sons
Teach Yourself REXX in 21 Days, 0-672-305291, Schindler & Schindler, Sams
McGraw-Hill, *Writing OS/2 REXX Programs*, 0-07-052372-X, Richardson
McGraw-Hill, *Writing VX-REXX Programs*, 0-07-9119-10, Richardson



New Products for OS/2

AM/Solo. This 4GL development tool for OS/2 maintains a CUA '91 library, including Notebook, Container and Slider Support. It provides extensive database support to DB2/2, DDCS/2, SQLServer, and Oracle and communication support for APPC, EHLLAPI, and Named Pipes. AM features full preemptive multitasking, online dictionaries, and full support of Warp flat memory models and 32-bit architecture for fast execution.

Intelligent Environments Circle No. 151
Phone: (800) 669-2797, (508) 640-1080, Fax:
(508) 640-1090

ATS 3.0 for OS/2. MHR Software and Consulting announces the release of Advanced Task Scheduler for OS/2 3.0. ATS is a production job scheduling utility that can be used by end users, developers, and on-network servers. Features include unlimited number of user-defined tasks, job queues to control resource utilization, events, and holidays; complete logging; API interface to all of the ATS features, U.S. or international date and time formats; and a console display of log, status, and running task information. ATS is a multithreaded, 32-bit OS/2 Presentation Manager application written entirely in C and compiled using IBM's C Set++ version 2.1.

MHR Software and Consulting Circle No. 152
Phone: (908) 821-0359, Fax: (908) 821-0350

CodeCheck 5.0. This software analysis tool allows automation of the C++ Quality Assurance Process. This technology will validate all source code produced as being in compliance or not. Individual programmers

having access to this product are able to correct their code before submission to the Quality Assurance Manager. CodeCheck reads all variants of C and C++ source code and is available for OS/2, Windows 95, and Macintosh. Source code for CodeCheck is available for license on all minicomputer and mainframe sites.

Abraxas Software Inc. Circle No. 153
Phone: (503) 244-5253, Fax: (503) 244-8375

Colorado Backup for OS/2. As a 32-bit application that takes advantage of OS/2's multitasking abilities, this product features five backup methods, custom file set recording, unattended scheduling, data compression, and command line operation. Supporting Jumbo and Trakker 120, 250, and 350 tape backup systems, it includes a two disk disaster recovery utility. It backs up FAT and HPFS systems and handles long filenames and extended attributes. Colorado Backup supports OS/2 2.1 and above, OS/2 Warp, and popular OS/2 networks.

Colorado Memory Systems Direct Circle No. 154
Phone: (303) 635-1500, Fax: (303) 663-7482

DataEase 5.0 for DOS and OS/2. This database software allows professional developers to build applications that present different views of the same database. Ultimately, people and departments view only the information they need, while filters ensure confidential data remains private. Developers can also place different parts of a database on different drives and directories. DataEase 5.0 for DOS and OS/2 and DataEase 5.0 for Windows are designed to share the same



database so people and departments throughout an organization can work together.

DataEase International Circle No. 155
Phone: (203) 374-8000, Fax: (203) 365-2317

DataTable 2.5. The Presentation Manager-based Spreadsheet/Grid Control is a development tool for OS/2. Within the operating system, the user can create complex data entry and display screens. The DataTable control offers several memory management schemes, including virtual memory management. With DataTable the user has the ability to display, manage, and edit rows, columns, and cells of data in a spreadsheet-like manner. The user can display database queries and multidimensional arrays of information, with over two billion rows and up to 250 columns. Features include setting fonts and colors for row and column labels, split windows, support for database null values, enhanced keyboard support, hidden columns, and faster column sorting with up to three sort keys.

ProtoView Development Co. Circle No. 156
Phone: (800) 231-8588, (908) 329-8588, Fax: (908) 329-8624

High C/C++ 3.3. This 32-bit global optimizing compiler for OS/2 works with IBM's System Object Model (SOM) to break the tight binary coupling and extend the advantages of procedure libraries to object-oriented technology. The breakthrough is achieved by a functionality called DirectToSOM, which enables developers to directly create and embed SOMobjects in the C++ programs. New features for 3.3 include fully implemented C++ exception handling, including nested exceptions; advanced namespace capabilities to avoid potential name clashes with third-party libraries; exclusive runtime-type information and cast notation; templates that track the draft ANSI C++ Standard; EasyThread functionality that generates code for thread-local storage; and programming support for IBM's Workplace Shell.

MetaWare Inc. Circle No. 157
Phone: (408) 429-6382, Fax: (408) 429-9273.

HLPDK/PA 12.5c. This is a multiplatform help authoring tool that supports the creation of

help files for multiple environments from one source file. The product includes converter programs from WinHelp, the Norton Guides, and PopHelp to HLPDK. With these converters, help files that were created on DOS and Windows environments can be translated to OS/2 and the Internet's World Wide Web.

HyperAct Inc. Circle No. 158
Phone: (319) 351-8413

Lotus Value Pack for OS/2. A collection of productivity tools, this package serves to enhance Lotus SmartSuite for OS/2 and the OS/2 Workplace Shell (WPS). This product improves users' day-to-day efficiency by tightening the integration between SmartSuite applications and the WPS, as well as among the SmartSuite applications themselves. The Value Pack for OS/2 comprises five components: SmartCenter for OS/2, Ami Pro Bonus Pack macros, the 1-2-3 File Translator, new Freelance SmartMaster presentation styles and business symbols, and REXXLink for 1-2-3.

Lotus Development Corp. Circle No. 159
Phone: (404) 828-5395

MKS RCS 5.2. This Revision Control System solves the problem of managing change during project development. It keeps a complete history of file changes and retrieves any version quickly and easily. Version 5.2 is a configuration management system that offers enhanced security and flexibility for distributed project development environments, plus a menu interface, binary and text file support, and supple configurability. It features an encryption facility for sensitive data, a user-configurable directory search path and filename template for RCS files, reverse delta storage, and full command line operation.

Mortice Kern Systems Inc./
Felder Computers Circle No. 160
(904) 778-2482

PartitionMagic for OS/2. This software utility permits users to visually move, shrink, expand, and convert their hard disk partitions. This product facilitates the process of creating space for a Boot Manager partition and automatically converts FAT partitions to HPFS. PartitionMagic for OS/2 includes a



32-bit DOS executable with a GUI to help DOS users modify their partitions.

PowerQuest Corp. Circle No. 161
Phone: (800) 379-2566 or (801) 226-8977, Fax: (801) 226-8941

PMCS-1000. Providing connection to a computer via the parallel port or SCSI port, PMCS-1000 is a two GB 3.5-in. minicartridge tape system housed in a portable minichassis with an internal auto-sensing power supply. It is a complete portable backup solution featuring parallel cable, SCSI cable, Quickstart User's Manual, PSS Software Package, and single-pass backup/verify operation. PCMS-1000 has a native storage capacity of 840 MB on a standard minicartridge and 1 GB on the new QIC-Wide Mini-Cartridge (included with the product) and compressed capacity of 1.7 GB and 2 GB, respectively. It provides for 8- to 18-MB/min. backup through a parallel port and up to 36 MB/min. through a SCSI port. The read-after-write capability of PMCS-1000 provides for single-pass backup/verify and restore/compare operations, saving time and guaranteeing data integrity.

Parallel Storage Solutions Circle No. 162
Phone: (800) 998-7839, (914) 347-7044, Fax: (914) 347-4646

ProMod. A system of CASE software development tools, ProMod is designed to run on open heterogeneous systems and supports popular design methodologies such as, Yourdon/DeMarco, McMenamin/Palmer, and Jackson Structured Analysis. With optional features, this product can be used for generating code (for C, FORTRAN, and C++), reverse engineering, schema generation, GUI design, and interactive simulation of system designs.

G&E Systems Inc. Circle No. 163
Phone: (800) 445-7899, (617) 784-1007, Fax: (617) 784-1737

Recore 3.0. Recore 3.0 offers an advanced set of integrated optical character recognition (OCR) software C-functions designed to create accurate and reliable OCR applications. This function library with a high-level program-

ming interface controls and manages the recognition operation, session control, and image and file registration. Recore 3.0's technical features include dual engine technology with Omnifont technology and training engine, Adaptive Intelligent Engine for improved efficiency, a modular command library, rule-based intelligent page segmentation providing the ability to read magazine and newspaper text while retaining the original layout, and zone data typing that enhances accuracy by configuring the software to read specific character formats on a zone by zone basis.

MAXSOFT-Ocron Inc. Circle No. 164
Phone: (800) 933-1399, (510) 252-0200, Fax: (510) 252-0202

Remote-OS. The new function for Remote-OS will provide access to DOS and OS/2 Text applications over TCP/IP connections. A license of PCP/IP for OS/2 from IBM is required to complete this new approach to Internet/Telnet application servers and network gateways. The combination of these products will allow terminals to be concurrently connected via serial ports and/or LAN adapter cards, providing access to real applications, such as cc:Mail or WordPerfect. These applications will run as if they were running on a terminal or Desktop PC.

The Software Lifeline Inc. Circle No. 165
Phone: (407) 994-4466, Fax: (407) 994-6304

Sytos Premium 2.1. This backup and recovery software offers OS/2 Warp users an intuitive Presentation Manager user interface that facilitates file selection and job execution. It includes Sytos Rebound, a disaster recovery utility that protects OS/2 Warp workstations and servers from total system failure. Designed to appeal to OS/2 Warp users, this product features a graphical user-friendly interface and automated operations. Sytos Premium has been updated to support additional command line switches, all network file and directory Access Control Lists, file permissions, and network system files. In addition, Sytos Rebound recovery utility will install on HPFS systems

using lowercase files. Sytos Premium provides optional autoloader device support with Sytos Autoloader 2.1 support module for OS/2.

Sytron Corp. Circle No. 166
Phone: (508) 898-0100, Automated fax response line: (508) 898-0001

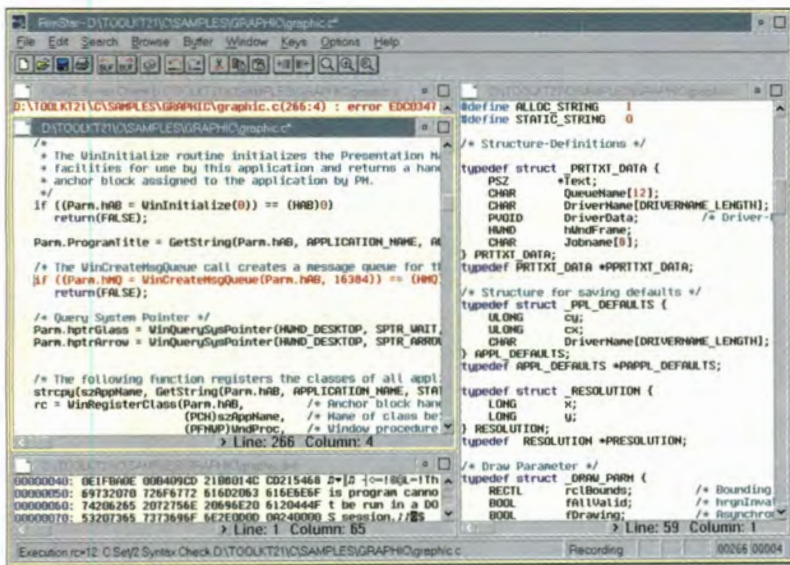
TechBridge Builder 1.2. The new release of this visual development tool includes master detail data display, use of Notebook control within a form, automated display of three-dimensional business graphics on a query form, and OS/2 Warp support. In addition, TechBridge has implemented an SQL-smart object framework that understands interfacing with the DB2 family. This includes transaction management, maintaining unit-of-work integrity, schema import, catalog fetch, business graphics and security interface, DDCCS interface, and composition of prepare and select statements. For the professional developer, there are complete APIs for direct data access and manipulation.

TechBridge Technology Corp. Circle No. 167
Phone: (800) 463-8998, (416) 222-8998, Fax: (416) 222-0168

X:Change. This fully graphical cross-platform management environment is specifically designed to meet the needs of application programmers that develop mainframe and client/server software on the workstation. Its 32-bit application provides programmers with technology to manage, transfer, and synchronize MVS and OS/2 files with point-and-click and drag-and-drop manipulation. X:Change can accommodate sequential, PDS(E), CAPANVALET, and CA-LIBRARIAN file structures on the mainframe and FAT, HPFS, and PVCS file formats on the PC. During transfer, it automatically converts EBCDIC to ASCII and vice versa. Also included is SmartCopy, a transfer technology that identifies duplicate versions of data set members or files on the target platform (even if they are named differently) and transfers only the nonduplicate versions.

Serena International Circle No. 168
Phone: (800) 457-3736, (415) 696-1800, Fax: (415) 696-1776

POWER UP WITH THE RIMSTARTM PROGRAMMER'S EDITOR NEW VERSION 2.1



If you're looking for a fully configurable, professional OS/2 PM programmer's editor to replace your current text-mode editor – we have what you have been looking for!

No hassle changing editors

We know changing editors can be a major hassle. You don't want to relearn a new set of keystrokes no matter how good the product. Well, you don't have to – we have Brief, CUA, Epsilon, Multi-Edit, SlickEdit, PWB, and Borland IDE keyboard mapping waiting for you. If you're not using one of these, let us know and we will create the mappings you need.

Features designed to increase productivity

It has all the features you have come to expect, plus special features designed to anticipate your needs and increase your productivity. Features like a 'C' source browser that eliminates those time consuming searches for functions, global variables, typedefs, and macros by providing you with instant positioning to both definitions and references. Using our powerful 'C' macro language you can customize your programming environment to suit your individual needs.

Partial Feature List

- ✓ Complete ANSI 'C' macro language
- ✓ SyntaColorTM—syntax highlighting
- ✓ Smart C/C++ indenting & brace matching
- ✓ Bookmarks
- ✓ Unlimited undo and redo
- ✓ Timed auto save
- ✓ Access PDK help for function under cursor
- ✓ Multi-threaded for no waiting
- ✓ Compile and jump to errors
- ✓ Import/export to system clipboard
- ✓ Keystroke record/playback
- ✓ Block indent/outdent
- ✓ Hex editing
- ✓ Customizable menus
- ✓ Column, line and block selection, search and replace
- ✓ Integrates with Workframe/2
- ✓ Source browser for 'C'
- ✓ Template editing
- ✓ Multi-buffer regular expression search and replace
- ✓ Support for version control
- ✓ Complete on-line help
- ✓ Save and restore state between sessions
- ✓ OS/2 2.x 32 bit PM Multi-document interface

"My copy of Brief has been permanently retired. Keep up the good work!" - A.L.

RimStar Technology, Inc.

91 Halls Mill Road
Newfields, NH 03856
Voice: (603) 778-2500
Fax: (603) 778-2408
BBS: (603) 778-4644

Price \$299.00

Plus Shipping & Handling.

To order call 1-800-746-7007

60 day money-back guarantee.

Also available for Windows and Windows NT
Circle Reader Service Number 37

All products and company names are trademarks or registered trademarks of their respective holders. RimStar and SyntaColor are trademarks of RimStar Technology, Inc.

© 1994 RimStar Technology Inc.

In the Race for Solutions, Finish First



Introducing VisPro/Reports, the first programmable report writer for VisPro/REXX, VX REXX and OS/2 REXX

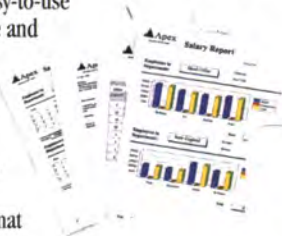
On April 28, 1993 HockWare Inc. shipped VisPro/REXX, the first OS/2 visual programming tool for REXX. Since then, we've achieved many other firsts. VisPro/REXX, VisPro/C and VisPro/C++ are the first REXX, C and C++ tools to incorporate drag and drop programming, an entity-relationship database designer, and a rich set of add-on objects including: spreadsheet, business graphics, formatted entry field, clock and calendar. Whether you're using REXX, C-Presentation Manager, or IBM's User-Interface Class Library, there is a VisPro product to meet your needs.

Yet Another First

VisPro/Reports is the latest addition to the VisPro family of OS/2 programming products. VisPro users will recognize the easy-to-use CUA'91 user-interface and the drag and drop style HockWare pioneered years ago.

Designing Reports is Easy

- WYSIWYG, free-format report design tool.



- Banded report options allow you to choose from a rich set of headers, footers and groups.
- Label reports allow you to select from a large list of Avery labels.
- Support for multiple line text, derived fields, business graphics, bitmap images, rectangles, lines and ellipses.
- Many object properties to choose from, including any OS/2 font, 16 million colors, line styles, fill patterns and drop shadows.
- Wide selection of field options, International currency and formulas.
- Numerous sample reports included.

Report Printing is Easy

- Seamlessly print or preview reports from any OS/2 REXX environment including VisPro/REXX, VX REXX and plain OS/2 REXX.
- Use your favorite REXX-enabled databases and third party libraries for access to DB2, .DBF files,

The Complete VisPro Family

VisPro/Reports	\$199
VisPro/REXX Gold	\$299
VisPro/REXX Bronze	\$99 \$59
VisPro/REXX Data Entry Object Pack	\$199 \$89
VisPro/C or VisPro/C++	\$399 \$199 special
VisPro Development Suite	\$599 \$499
(includes VisPro/REXX Gold, C and C++)	

Whether you've already discovered the VisPro family of visual programming products or you're just getting to know us, meet our latest member, VisPro/Reports. And be the first to get where you're going.

Special Offer
Buy
VisPro/REXX Gold
for \$299 and get
VisPro/Reports
absolutely free.
Offer Expires 6/15/95

6362-0001-29



PRINTED IN USA



HockWare Incorporated
P.O. Box 336
Cary, NC 27512-0336
919-380-0616
919-380-0757 FAX
Go HockWare on CompuServe
hockware@vnet.net on Internet

HockWare, VisPro/C, VisPro/C++, VisPro/REXX and VisPro/Re are trademarks of HockWare Incorporated. All other company, products and brand names are trademarks and/or registered trademarks of their respective holders and are mentioned for reference purposes only. © 1995 HockWare Incorporated. All rights reserved.

Circle Reader Service Number 38